Библиотека системного программиста

ЛОКАЛЬНЫЕ СЕТИ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Использование протоколов IPX, SPX, NETBIOS

ДИАЛОГ-МИФИ

БИБЛИОТЕКА СИСТЕМНОГО ПРОГРАММИСТА
ВОСЬМОЙ ТОМ

А. В. ФРОЛОВ, Г. В. ФРОЛОВ

ЛОКАЛЬНЫЕ СЕТИ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Использование протоколов IPX, SPX, NETBIOS

МОСКВА • "ДИАЛОГ-МИФИ" • 1993

ББК 32.973.1
Ф91

БИБЛИОТЕКА
СИСТЕМНОГО ПРОГРАММИСТА
Выпускается с 1991 года

Фролов А. В., Фролов Г. В.
Ф91 Локальные сети персональных компьютеров. Использование протоколов IPX, SPX, NETBIOS. — М.: "ДИАЛОГ-МИФИ", 1993. — 160 с. — (Библиотека системного программиста; Т. 8)

ISBN 5-86404-033-9 (Т. 8)

В книге рассказывается об использовании протоколов IPX, SPX и NETBIOS в программах, предназначенных для работы в среде оболочек рабочих станций локальных сетей Novell NetWare. Приведены описания протоколов, примеры программ на языках ассемблера и Borland C++, а также другие сведения, необходимые для обеспечения корректной работы программ в локальных сетях персональных компьютеров, совместимых с IBM PC.

Ф 2404070000-006 / Г70(03)-93 Без объявл. ББК 32.973.1

Учебно-справочное издание
Фролов Александр Вячеславович
Фролов Григорий Вячеславович
Локальные сети персональных компьютеров.
Использование протоколов IPX, SPX, NETBIOS

Редактор О.А. Кузьминова
Макет и обложка Н.В. Дмитриевой
Корректор В.С. Кустов

Лицензия ЛР N 070109 от 29.08.91. Подписано в печать 3.11.93. Формат 60x84/16. Бум. офс. Печать офс. Гарнитура Таймс. Усл. печ. л. 9.30. Уч.-изд. л. 6.82. Тираж 20 000 экз. Заказ 1085

Акционерное общество "ДИАЛОГ-МИФИ"
115409, Москва, ул. Москворечье, 31, корп. 2

Подольская типография

142100, г. Подольск, Московская обл., ул. Кирова, 25

ISBN 5-86404-033-9 (Т. 8)
ISBN 5-86404-004-5

ВВЕДЕНИЕ

В предыдущем томе "Библиотеки системиого программиста" мы иаучили вас устанавливать аппаратное и программиое обеспечеиие для самых распространеиных сетевых операциоииых систем - Novell NetWare версий 2.2 и 3.11, Novell NetWare Lite, а также Microsoft Windows for Workgroups. Надеемся, что у вас все получилось и вы имеете возможиость работать в сети (а главиое, что такую возможиость имеют все сотрудиики вашей организации).

Следующий этап - программирование для локальных сетей. Под программированием для локальных сетей понимается составление программ, предназиаченных для работы как в составе сетевых операционных систем, так и в составе программного обеспечеиия рабочих станций сети. Сразу отметим, что в этой книге мы ие будем рассматривать вопросы, связаниые с созданием программного обеспечения, работающего иа файл-серверах Novell NetWare (nlm- и vap-программы), а ограничимся лишь сетевым программным обеспечением рабочих станций. На первом этапе такое ограничение ие играет существенной роли - большинство достаточио сложных проблем можно решить в рамках операционных систем рабочих станций, ие прибегая к программированию в среде Novell NetWare.

Используя сведеиия, приведеииые в иашей кииге, вы сможете решить такие задачи, как организация связи между программами, работающими иа разиых стаициях в сети без обращеиия к файл-серверу, создание собствеииых серверов для работы с модемами или базами даиных, разработка электроиной почты, разработка игровых программ для сети и миогое другое.

Что вам поиадобится для работы с киигой?

Во-первых, безусловио, нужна работающая локальная сеть, даже если в ией всего два компьютера. Лучше всего, если это будет сеть Novell NetWare версии 3.11 или 2.2, однако с протоколом IPX вы сможете работать и в сети NetWare Lite, причем вам будет достаточио установить сетевые оболочки иа рабочие станции. Для работы с протоколом IPX или SPX сервер NetWare вообще ие нужеи!

Во-вторых, вам потребуется среда разработки программ Borland C++ версии 3.1 (во всяком случае, все программы, приведеииые в киите, отлаживались имеиио в этой среде). При соответствующей адаптации вы сможете воспользоваться Borland C++ версии 3.0 или даже 2.0. Те программы, в которых ие используется объектно-ориеитироваииый подход, могут с иекоторыми миимальиыми исправлеииями траислироваться в среде Microsoft Quick C версии 2.51 или Microsoft C версии 6.0.

Глава 1

ПЕРЕДАЧА ДАННЫХ В ЛОКАЛЬНОЙ СЕТИ

В локальной сети данные передаются от одиой рабочей станции к другой блоками, которые иазывают пакетамн данных. Каждый пакет состоит из заголовка и собственно блока данных. Станция, которая желает передать пакет данных другой станции, указывает в заголовке адрес назначения и свой собственный, аналогично тому, как это делаете вы, отправляя обычное пнсьмо. На конверте, в который вложеио пнсьмо, вы указываете адрес получателя н обратный (свой собственный) адрес.

Продолжая аналогию с письмами, вспомним, что иа почте существует такая услуга, как отправка письма или телеграммы с уведомлением о вручении. Когда адресат получит ваше письмо, вам отправляется уведомление о вручении. В этом случае вы можете убедиться, что письмо дошло до адресата и не потерялось по дороге.

В локальиой сети программы также имеют возможность отправлять "обычные письма", а также "письма с уведомлеиием о врученнн". И, разумеется, в локальной сети имеется своя система адресов.

1.1. Датаграммы

Передача пакетов даниых между рабочимн стаицiями без подтверждеиия - это тип связи между рабочими станциями иа уровие датаграмм (datagram). Уровеиь датаграмм соответствует сетевому уровню (Network Layer) семиуровневой модели OSI, опнсаиной нами в предыдущем томе.

Что зиачит "передача без подтверждеиия"? Это означает, что ие гарантируется доставка пакета от передающей станцни к принимающей. В результате, например, перегрузки сети или по какнм-либо другим причниам принимающая сторона может так и не дождаться предиазиачеииого ей пакета даниых. Причем, что характерно для уровня датаграмм, передающая сторона так и не узиает, получила ли принимающая сторона пакет или ие получила.

Более того, иа уровне датаграмм ие гарантируется также, что принимающая сторона получит пакеты в той последовательиости, в какой оии посылаются передающей стаицией!

Казалось бы, зачем нужна такая передача даниых, которая ие гарантирует доставки? Однако программы, обменивающиеся даиными, могут самн организовать проверку. Напрнмер, прниимающая программа может сама посылать подтвержденне передающей программе о том, что получеи пакет данных.

Протокол передачи даииых IPX - межсетевой протокол передачи пакетов (Internetwork Packet Exchange) - используется в сетевом программном обеспечении Novell н является реализацией уровня датаграмм. Протокол NETBIOS, разработаниый фирмой IBM, также может работать иа уровне датаграмм.

Большиистве задач в сети можно решить иа уровие датаграмм, поэтому мы уделим миого виимаиия протоколам IPX и NETBIOS.

Одно из преимуществ уровня датаграмм - возможность посылки пакетов данных одновременно всем станциям в сети. Если же для программ необходима гарантированная доставка данных, можно использовать протокол более высокого уровня - уровня сеанса связи.

1.2. Сеансы связи

На уровне сеансов связи (Session Layer) две рабочие станции перед началом обмена данными устанавливают между собой канал связи - обмениваются пакетами специального вида. После этого начинается обмен данными.

На уровне сеансов связи при необходимости выполняются повторные передачи пакетов данных, которые по каким-либо причинам "не дошли" до адресата. Кроме того, гарантируется, что принимающая станция получит пакеты данных именно в том порядке, в котором они были переданы.

При использовании уровня сеансов связи невозможно организовать "широковещательную" передачу пакетов одновременно всем станциям - для передачи данных необходимо организовать канал связи между одной и другой станцией. Следовательно, в процессе передачи данных могут участвовать одновременно только две станции.

Как и следовало ожидать, в сетевом программном обеспечении Novell уровень сеансов связи реализован как надстройка над уровнем датаграмм. На базе протокола IPX реализован протокол SPX - протокол последовательной передачи пакетов (Sequenced Packet Exchange Protocol).

Протокол NETBIOS реализует наряду с уровнем датаграмм уровень сеансов связи.

В сети Novell NetWare есть эмулятор протокола NETBIOS. Этот эмулятор использует протокол IPX для реализации как уровня датаграмм, так и уровня сеансов связи.

1.3. Сетевой адрес

Подобно почтовому адресу, сетевой адрес состоит из нескольких компонентов. Это номер сети, адрес станции в сети и идентификатор программы на рабочей станции - сокет (рис. 1).

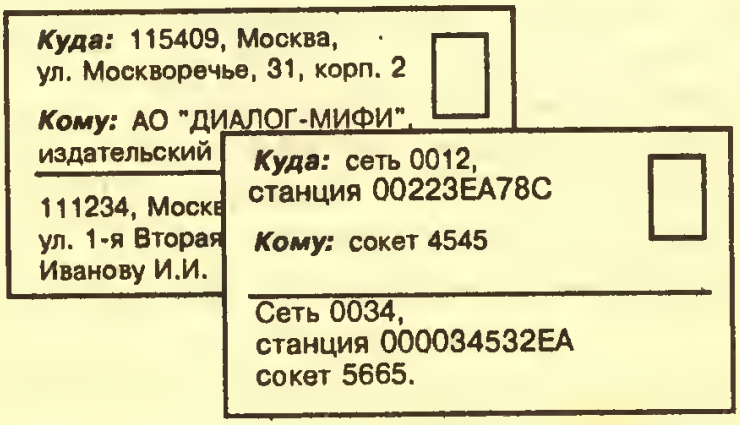


Рис. 1. Сетевой адрес

Номер сети (network number) - это номер сегмента сетн (кабельного хозяйства), определяемого системным администратором при установке Novell NetWare. Не путайте этот номер с внутреннim номером сети файл-сервера. Напомнim, что еслн в одном сегменте сети имеется два файл-сервера NetWare, то они оба имеют одинаковый номер сетн, но разные внутренние номера сети. Еслн в общей сети есть мосты, каждая отдельная сеть, подключенная через мост, должна иметь свой, уникальный номер сетн.

Адрес станции (node address) - это число, которое является уннкальным для каждой рабочей станцнн. При использованнн адаптеров Ethernet уникальность обеспечивается изготовителем сетевого адаптера (адрес станцин запнсан в мнкросхеме постоянного запомннающего устройства, которая находится внутри самого адаптера). Для адаптеров ArcNet адрес станции необходимо устанавливать при помощи перемычек или переключателей на плате сетевого адаптера. Устанавливая в сетн адаптеры ArcNet, позаботьтесь о том, чтобы все они нмели в сети разные адреса. Как установить сетевой адрес адаптера ArcNet, вы сможете узнать из документацин, поставляющейся вместе с адаптером.

Специальный адрес FFFFFFFFFFFFh используется для посылки пакета данных всем станцням данной сетн одновременно. Пакет с такнм адресом напоминает открытое письмо с опубликованием в печати.

Идентификатор программы на рабочей станции - сокет (socket) - число, которое используется для адресацнн конкретной программы, работающей на станции. В среде мультизадачных операционных систем, к которым можно отнести OS/2 н Microsoft Windows в расшренном режиме, на каждой рабочей станции в сети одновременно могут быть запущены несколько программ. Для того, чтобы послать данные конкретной программе, используется идентификация программ прн помощн сокетов. Каждая программа, желающая принимать илн передавать данные по сети, должна получить свой, уникальный для данной рабочей станции, идентификатор - сокет.

Глава 2

ПРОТОКОЛ IPX

Протокол IPX предоставляет возможиость программам, запущениым на рабочих станциях, обменнваться пакетами даиных на уровне датаграмм, т. е. без подтверждеиия.

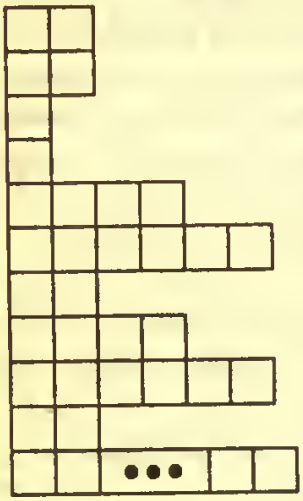
В сети Novell NetWare ианболее быстрая передача данных прн иаиболее экономном использовании памяти реализуется имеиио протоколом IPX. Протоколы SPX н NETBIOS сделаиы иа базе IPX и поэтому требуют дополиительных ресурсов. Поэтому мы иачием изучеиие программироваиия для локальиых сетей нменно с протокола IPX.

Если на рабочей стаицни нспользуется операцноииая система MS-DOS, функцни, необходнмые для реалнзацнн протокола IPX, реализуются резидентиымн программами ipx.com илн ipxodi.com, входящимн в состав сетевой оболочкн рабочей станцни сети NetWare.

Для того чтобы иаучиться составлять программы, которые могут передавать даиные по сети с нспользованнем протокола IPX, вам необходнмо познакомиться со структурой пакета IPX (вам придется создавать такне пакеты) н иаучнться пользоваться функциями IPX, реалнзованиымн в рамках сетевой оболочки рабочей стаицнн. Вначале мы позиакомим вас со структурой пакета IPX, затем займемся фуикциямн.

2.1. Формат пакета IPX

Формат передаваемых по сети пакетов представлеи иа рнс. 2.



Checksum - контрольная сумма
Length - общая длина пакета
TransportControl - счетчик пройденных мостов
PacketType - тип пакета
DestNetwork - номер сети получателя пакета
DestNode - адрес станции-получателя
DestSocket - сокет программы-получателя
SourceNetwork - номер сети отправителя пакета
SourceNode - адрес станции-отправителя
SourceSocket - сокет программы-отправителя
Data - передаваемые данные, от 0 до 546 байт

Рис. 2. Структура пакета IPX

Пакет можно разделить иа две части - заголовок и передаваемые данные. Все поля, представленные иа рис. 2, кроме последнего (Data), представляют собой заголовок

пакета. Заголовок пакета выполняет ту же роль, что и конверт обычного письма - там располагается адрес назначения, обратный адрес и некоторая служебная информация.

Особенностью формата пакета является то, что все поля заголовка содержат значения в перевернутом формате, т. е. по младшему адресу записывается старший байт данных, а не младший, как это принято в процессорах фирмы Intel. Поэтому перед записью значений в многобайтовые поля заголовка необходимо выполнить соответствующее преобразование. Представление данных в заголовке пакета соответствует, например, формату целых числел в компьютере IBM-370 (серия ЕС ЭВМ).

Рассмотрим подробнее назначение отдельных полей пакета.

Поле **Checksum** предназначено для хранения контрольной суммы передаваемых пакетов. При формировании собственных пакетов вам не придется заботиться о содержимом этого поля, так как проверка данных по контрольной сумме выполняется драйвером сетевого адаптера.

Поле **Length** определяет общий размер пакета вместе с заголовком. Длина заголовка фиксирована и составляет 30 байт. Размер передаваемых в поле Data данных может составлять от 0 до 546 байт, следовательно, в поле Length в зависимости от размера поля Data могут находиться значения от 30 до 576 байт. Если длина поля Data равна нулю, пакет состоит из одного заголовка. Как это ни странно, такие пакеты тоже нужны! При формировании собственных пакетов вам не надо проставлять длину пакета в поле Length, протокол IPX сделает это сам (вернее, программный модуль, отвечающий за реализацию протокола IPX, вычислит длину пакета на основании длины поля Data).

Поле **TransportControl** служит как бы счетчиком мостов, которые проходит пакет на своем пути от передающей станции к принимающей. Каждый раз, когда пакет проходит через мост, значение этого счетчика увеличивается на единицу. Перед передачей пакета IPX сбрасывает содержимое этого поля в нуль. Так как IPX сам следит за содержимым этого поля, при формировании собственных пакетов вам не надо изменять или устанавливать это поле в какое-либо состояние.

Поле **PacketType** определяет тип передаваемого пакета. Программа, которая передает пакеты средствами IPX, должна установить в поле PacketType значение 4. Протокол SPX, реализованный на базе IPX, использует в этом поле значение 5.

Поле **DestNetwork** определяет номер сети, в которую передается пакет. При формировании собственного пакета вам необходимо заполнить это четырехбайтовое поле. Напомним, что номер сети задается сетевым администратором при установке Novell NetWare на сервер.

Поле **DestNode** определяет адрес рабочей станции, которой предназначен пакет. Вам необходимо определить все шесть байт этого поля.

Поле **DestSocket** предназначено для адресации программы, запущенной на рабочей станции, которая должна принять пакет. При формировании пакета вам необходимо заполнить это поле.

Поля SourceNetwork, SourceNode и SourceSocket содержат соответствеино иомер сети, из которой посылается пакет, адрес передающей станции и сокет программы, передающей пакет.

Поле Data в пакете IPX содержит передаваемые данные. Как мы уже говорили, длина этого поля может быть от 0 до 546 байт. Если длиа поля Data равиа нулю, пакет состоит из одного заголовка. Такой пакет может использоваться программой, например, для подтверждения приема пакета с данными.

Для формироваиия заголовка пакета можно воспользоваться, например, следующей структурой:

```
struct _IPXHeader {
   unsigned char Checksum[2];
   unsigned char Length[2];
   unsigned char TransportControl;
   unsigned char PacketType;
   unsigned char DestNetwork[4];
   unsigned char DestNode[6];
   unsigned char DestSocket[2];
   unsigned char SourceNetwork[4];
   unsigned char SourceNode[6];
   unsigned char SourceSocket[2];
} IPXHeader;
```

Обращаем ваше внимание иа то, что все многобайтовые поля описаны как массивы. Даже те, которые состоят из двух байт и могли бы быть описаны как unsigned int. Это связано с тем, что все зиачения в заголовке пакета IPX хранятся в перевернутом виде, а для такого типа данных в языке Си иет подходящего описания.

2.2. Работа с драйвером IPX/SPX

Первое, что должиа сделать программа, желающая работать в сети с протоколом IPX или SPX, - проверить, установлен ли драйвер соответствующего протокола. Затем необходимо получить адрес вызова этого драйвера - точку входа API (Application Programm Interface - интерфейс для приложений). В дальиейшем программа вызывает драйвер при помощи команды межсегментного вызова процедуры по адресу точки входа API драйвера IPX/SPX.

2.2.1. Точка входа API драйвера IPX/SPX

Для того чтобы проверить, загружен ли драйвер IPX, необходимо загрузить в регистр AX зиачение 7A00h и вызвать мультиплексное прерываиие INT 2Fh.

Если после возврата из прерываиия в регистре AL будет значеиие FFh, драйвер IPX загружен. Адрес точки входа для вызова API драйвера при этом будет находиться в регистровой паре ES:DI.

Если же содержимое регистра AL после возврата из прерываиия INT 2Fh будет отличио от FFh, драйвер IPX/SPX не загружен. Это озиачает, что на даиной

рабочей станции не загружены резидентные программы **ipx.exe** или **ipxodi.exe**, обеспечивающие API для работы с протоколами IPX и SPX.

Для вызова API в регистр BX необходимо загрузить код выполняемой функции. Значения, загружаемые в другие регистры, зависят от выполняемой функции.

Например, функция с кодом 10h используется для проверки присутствия в системе протокола SPX (может быть такая ситуация, когда протокол IPX присутствует, а SPX - нет). Для того, чтобы определить наличие SPX, необходимо загрузить в BX значение 10h, в AX значение 00h и вызвать API драйвера IPX. Если после возврата регистр AX будет содержать значение FFh, протокол SPX присутствует и может быть использован. В регистрах CX и DX передаются параметры SPX - максимальное число каналов связи, которое данная станция может установить с программами, работающими на других станциях, и количество каналов, доступное в настоящее время. О назначении этих параметров мы будем говорить в главе, посвященной протоколу SPX.

Приведем текст программы, определяющей наличие драйвера протоколов IPX и SPX (листинг 1). Программа вызывает функции ipx_init() и ipxspx_entry(), тексты которых находятся в листинге 2. Текст сокращенного варианта include-файла ipx.h представлен в листинге 3.

Вы можете попробовать запустить эту программу на рабочей станции сети Novell NetWare под управлением MS-DOS или на виртуальной машине MS Windows, работающей в расширенном (Enchanced) режиме.

```
// ================================================
// Листинг 1. Программа для обнаружения драйвера
// протокола IPX/SPX и определения его версии
//
// Файл ipxver.c
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include "ipx.h"
void main(void) {
// Точка входа в IPX/SPX API, переменная находится
// в файле ipxdrv.asm и инициализируется функцией ipx_init().
     extern far char *ipxspx_drv_entry;
// Структура для вызова API IPX
     struct IPXSPX_REGS iregs;
     unsigned error;
     unsigned spx_ver;
     unsigned spx_max_connections, spx_avail_connections;
     printf("\n*Детектор IPX/SPX*, (C) Фролов А., 1993\n\n");
// Проверяем наличие драйвера IPX и определяем
```

```
// адрес точки входа его API
     if(ipx_init() == 0xff) printf("IPX загружен! ");
     else {
          printf("IPX не загружен!\n"); exit(-1);
     }
     printf("Точка входа в IPX API - %Fp\n",ipxspx_drv_entry);
// Проверяем доступность протокола SPX
          error = NO_ERRORS;
// Вызываем функцию проверки доступности SPX
// Здесь мы вызываем API драйвера IPX/SPX
          iregs.bx = SPX_CMD_INSTALL_CHECK;
          iregs.ax = 0;
          ipxspx_entry( (void far *)&iregs );
          if(iregs.ax == 0x00) error = ERR_NO_SPX;
          if(iregs.ax != 0xff) error = UNKNOWN_ERROR;
          if(error != NO_ERRORS) {
               printf("SPX не загружен!\n"); exit(-1);
          }
// Запоминаем параметры IPX/SPX
          spx_ver = iregs.bx;
          spx_max_connections = iregs.cx;
          spx_avail_connections = iregs.dx;
          printf("SPX загружен! ");
          printf("Версия SPX: %d.%d\n", (spx_ver>>8) & 0xff,
                                          spx_ver & 0xff);
          printf("Всего соединений: %d, ", spx_max_connections);
          printf("из них доступно: %d\n", spx_avail_connections);
          exit(0);
}
```

Далее расположен исходный текст модуля инициализации IPX (листинг 2).

В этом модуле находится функция ipxspx_entry(), необходимая для вызова драйвера IPX/SPX. Ее имя начинается с символа "_", что необходимо для выполнения соглашения об именах в языке Си.

Здесь же имеется функция ipx_init(), которая проверяет наличие драйвера в системе, получает адрес API драйвера и сохраняет его в области памяти _ipxspx_drv_entry.

```
; ================================================
; Листинг 2. Инициализация и вызов драйвера IPX/SPX
; Файл ipxdrv.asm
;
; (C) A. Frolov, 1993
; ================================================
.286
.MODEL SMALL
```

```
; ------------------------------------
; Структура для вызова драйвера IPX/SPX
; ------------------------------------
     IPXSPX_REGS struc
          rax   dw ?
          rbx   dw   ?
          rcx   dw   ?
          rdx   dw   ?
          rsi   dw   ?
          rdi   dw   ?
          res   dw   ?
     IPXSPX_REGS ends
.DATA
; Точка входа в драйвер IPX/SPX
_ipxspx_drv_entry     dd ?
.CODE
     PUBLIC     _ipxspx_entry, _ipx_init
     PUBLIC     _ipxspx_drv_entry
; ------------------------------------
; Процедура, вызывающая драйвер IPX/SPX
; ------------------------------------
_ipxspx_entry PROC FAR
; Готовим BP для адресации параметра функции
          push bp
          mov  bp,sp
; Сохраняем регистры, так как драйвер IPX/SPX
; изменяет содержимое практически всех регистров
          push es
          push di
          push si
          push dx
          push cx
          push bx
          push ax
; Загружаем регистры из структуры,
; адрес которой передается как параметр
          push ds
          mov  bx, [bp+6]   ; смещение
          mov  ds, [bp+8]   ; сегмент
          mov  es, ds:[bx].res
          mov  di, ds:[bx].rdi
          mov  si, ds:[bx].rsi
          mov  dx, ds:[bx].rdx
          mov  cx, ds:[bx].rcx
```

```
            mov   ax, ds:[bx].rax
            mov   bx, ds:[bx].rbx
            pop   ds
; Вызываем драйвер IPX/SPX
            call  [dword ptr _ipxspx_drv_entry]
; Сохраняем регистры
            push  ds
            push  dx
            mov   dx, bx
; Записываем в структуру содержимое регистров после вызова драйвера
            mov   bx, [bp+6]  ; смещение
            mov   ds, [bp+8]   ; сегмент
            mov   ds:[bx].rax, ax
            mov   ds:[bx].rcx, cx
            mov   ds:[bx].rbx, dx
            pop   dx
            mov   ds:[bx].rdx, dx
            pop   ds
; Восстанавливаем регистры
            pop   ax
            pop   bx
            pop   cx
            pop   dx
            pop   si
            pop   di
            pop   es
            pop   bp
            retf
_ipxspx_entry ENDP
; ------------------------------------------
; Процедура инициализации драйвера IPX/SPX
; ------------------------------------------
_ipx_init PROC NEAR
            push  bp
            mov   bp,sp
; Определяем наличие драйвера в системе и его точку входа
            mov   ax, 7a00h
            int   2fh
; Если драйвера нет, завершаем процедуру
            cmp   al, 0ffh
            jne   _ipx_init_exit
; Сохраняем адрес точки входа
            mov   word ptr _ipxspx_drv_entry+2, es
```

```
            mov   word ptr _ipxspx_drv_entry, di
_ipx_init_exit:
; В регистре AX - код завершения процедуры
            mov   ah, 0
            pop   bp
            ret
_ipx_init ENDP
end
```

Описания типов и констант, а также прототипы функций для программы ipxver.c находятся в файле ipx.h (листинг 3).

```
// ================================================
// Листинг 3. Include-файл для работы с IPX
// Сокращенный вариант для программы ipxver.c
// Файл ipx.h
//
// (C) A. Frolov, 1993
// ================================================
// ----------------------
// Команды интерфейса IPX
// ----------------------
#define IPX_CMD_OPEN_SOCKET                0x00
#define IPX_CMD_CLOSE_SOCKET               0x01
#define IPX_CMD_GET_LOCAL_TARGET           0x02
#define IPX_CMD_SEND_PACKET                0x03
#define IPX_CMD_LISTEN_FOR_PACKET          0x04
#define IPX_CMD_SCHEDULE_IPX_EVENT         0x05
#define IPX_CMD_CANCEL_EVENT               0x06
#define IPX_CMD_GET_INTERVAL_MARKER        0x08
#define IPX_CMD_GET_INTERNETWORK_ADDRESS   0x09
#define IPX_CMD_RELINQUISH_CONTROL         0x0a
#define IPX_CMD_DISCONNECT_FROM_TARGET     0x0b
// ----------------------
// Команды интерфейса SPX
// ----------------------
#define SPX_CMD_INSTALL_CHECK              0x10
// ----------------------
// Коды ошибок
// ----------------------
#define NO_ERRORS          0
#define ERR_NO_IPX         1
#define ERR_NO_SPX         2
#define NO_LOGGED_ON       3
#define UNKNOWN_ERROR      0xff
```

```
// ----------------------
// Константы
// ----------------------
#define SHORT_LIVED   0
#define LONG_LIVED    0xff
#define IPX_DATA_PACKET_MAXSIZE 546
// Внешние процедуры для инициализации и вызова драйвера IPX/SPX
void far     ipxspx_entry(void far *ptr);
int          ipx_init(void);
// Структура для вызова драйвера IPX/SPX
struct IPXSPX_REGS {
          unsigned int    ax;
          unsigned int    bx;
          unsigned int    cx;
          unsigned int    dx;
          unsigned int    si;
          unsigned int    di;
          unsigned int    es;
};
```

2.2.2. Использоваиие API драйвера IPX

Теперь, после того как мы иаучились проверять иаличие драйвера IPX и определять точку входа для вызова его API, иам предстоит иаучиться пользоваться этим API. Без преувеличеиия можио сказать, что от того, иасколько хорошо вы освоите API драйвера IPX, зависят ваши успехи в создании программиого обеспечеиия для сетей Novell NetWare.

Однако, прежде чем мы займемся программным интерфейсом IPX, иам необходимо разобраться со структурой программ, способных обмениваться данными по сети, и понять, каким образом программы, работающие иа различных станциях, могут передавать друг другу эти самые данные. Такие зиания помогут вам и в том случае, если вы будете создавать программы, использующие протокол NETBIOS.

Схема "клиент–сервер"

Обычио в сети одиа из рабочих стаиций прииимает запросы иа выполиеиие каких-либо действий от других рабочих стаиций. Так как стаиция обслуживает запросы, оиа иазывается сервером (serve - обслуживать, server - обслуживающее устройство). Выполиив запрос, сервер посылает ответ в запросившую стаицию, которая иазывается клиеитом.

В сети может быть миого серверов и миого клиентов. Одии и те же клиенты могут посылать запросы разиым серверам.

Говоря более строго, сервером или клиентом является не рабочая станция, а запущенная на ней программа. В мультизадачной среде разные программы, запущенные одновременно на одной и той же рабочей станции могут являться и клиентами, и серверами.

Программа-сервер, выполнив очередной запрос, переходит в состояние ожидания. Она ждет прихода пакета данных от программы-клиента. В зависимости от содержимого этого пакета программа-сервер может выполнять различные действия, в соответствии с логикой работы программы. Например, она может принять от программы-клиента дополнительные пакеты данных или передать свои пакеты.

Сервер и клиент при необходимости на какое-то время или навсегда могут поменяться местами, изменив свое назначение на противоположное.

Для того, чтобы создавать программы-серверы и программы-клиенты, нам необходимо научиться выполнять две задачи:

- инициализацию сервера и клиента;
- прием и передачу пакетов данных.

Инициализация сервера и клиента

Для инициализации программ сервера и клиента, работающих на базе IPX, недостаточно убедиться в наличии соответствующего драйвера и получить точку входа в его API. Вам необходимо выполнить некоторые подготовительные действия для того, чтобы программа могла принимать и передавать пакеты данных.

Прежде всего необходимо, чтобы программа-сервер или программа-клиент идентифицировали себя в сети при помощи механизма сокетов.

Для хранения сокета используется двухбайтовое слово, так что диапазон возможных значений простирается от 0 до FFFFh. Однако вы не можете использовать произвольные значения.

Некоторые значения зарезервированы для использования определенными программами. Это так называемые "хорошо известные" сокеты ("well-known" sockets).

Так как протокол IPX является практической реализацией протокола Xerox Internetwork Packet Protocol, первоначальное распределение сокетов выполняется фирмой Xerox. Согласно этому распределению сокеты от 0 до 3000 зарезервированы статически за определенным программным обеспечением. В частности, фирма Novell получила от фирмы Xerox диапазон сокетов для своей сетевой операционной системы NetWare. В спецификации Xerox сокеты со значением, большим чем 3000, могут распределяться динамически.

Динамически распределяемые сокеты выдаются программам как бы во временное пользование (на время их работы) по специальному запросу. Перед началом работы программа должна запросить сокет у протокола IPX, а перед завершением - освободить его.

Распределение сокетов в сети Novell NetWare несколько отличается от распределения, установленного фирмой Xerox. Сокеты от 0 до 4000h зарезервированы и не должны использоваться в программном обеспечении пользователей. Сокеты от 4000h до 8000h распределяются динамически. Диапазон "хорошо известных" сокетов, распределяемых Novell персонально разработчикам программного обеспечения, расположен выше значения 8000h.

Вы, как разработчик программного обеспечения для сетей NetWare, можете получить у Novell для своей программы персональный сокет (если сумеете это

сделать) или воспользоваться сокетом, полученным динамически. Можно задавать сокет в качестве параметра при запуске программы. Если вы обнаружите, что используемое вами значение сокета конфликтует с другим программным обеспечением, вы легко сможете изменить его, просто задавая новое значение для соответствующего параметра.

При реализации схемы обмена данными "клиент–сервер" сервер обычно принимает пакеты на сокете, значение которого известно программам-клиентам. Сами же программы-клиенты могут использовать либо то же самое значение сокета, либо получать свой сокет динамически. Клиент может сообщить серверу свой сокет просто передав его в пакете данных (так как мы предполагаем, что сокет сервера известен программе-клиенту).

После определения сокета необходимо узнать сетевой адрес станций-получателей. Для того чтобы клиент мог послать запрос серверу, необходимо кроме сокета сервера знать его сетевой адрес - номер сети и адрес рабочей станции в сети.

Если программа-клиент знает только сокет программы-сервера, но не знает его сетевой адрес, последний можно запросить у сервера, послав запрос во все станции одновременно. Такой запрос в пределах одного сегмента сети можно выполнить, если в качестве адреса рабочей станции указать специальное значение FFFFFFFFFFFFh. Это так называемый "широковещательный" (broadcast) адрес.

Клиент посылает запрос на известный ему сокет программы-сервера и использует адрес FFFFFFFFFFFFh. Такой запрос принимают все программы на всех рабочих станциях, ожидающие пакеты на данном сокете. Получит его и наша программа-сервер. А она может определить свой собственный сетевой адрес (выполнив вызов соответствующей функции IPX) и послать его клиенту. Адрес же клиента программа-сервер может взять из заголовка принятого пакета.

Разумеется, существует способ определения адреса рабочей станции по имени пользователя, подключившегося на ней к файл-серверу. Это можно сделать при помощи API сетевой оболочки рабочей станции (резидентная программа netx.exe). Однако этот способ не позволит вам определить адрес станции, на которой не выполнено подключение к файл-серверу или не запущена сетевая оболочка netx.exe. Пакет, переданный по адресу FFFFFFFFFFFFh, будет принят всеми станциями сети даже в том случае, если файл-сервер выключен или его вовсе нет. Поэтому способ определения сетевого адреса через запрос по всей сети более универсален.

Прием и передача пакетов данных

Рассмотрим теперь процедуру приема пакетов данных средствами IPX.

Прием и передачу пакетов выполняет сетевой адаптер, работающий с использованием прерываний. Некоторые сетевые адаптеры работают с памятью через канал прямого доступа DMA. Прерывание от сетевого адаптера обрабатывает драйвер сетевого адаптера. Например, в операционной системе MS-DOS для адаптеров, совместимых с адаптером Novell NE2000 в составе Novell NetWare поставляется драйвер ne2000.com, реализованный в виде резидентной программы.

Прикладные программы не работают напрямую с драйвером сетевого адаптера. Все свои запросы на прием и передачу пакетов они направляют драйверу IPX (программа ipx.exe или ipxodi.exe), который, в свою очередь, обращается к драйверу сетевого адаптера.

Для приема илн передачи пакета прикладная программа должна подготовнть пакет данных, сформировав его заголовок, н построить так называемый блок управления событием ECB (Event Control Block). В блоке ECB задается адресная информация для передачн пакета, адрес самого передаваемого пакета в оперативной памяти и некоторая другая информация.

Подготовив блок ECB, прикладная программа передает его адрес соответствующей функцнн IPX для выполненія операции прнема или передачн пакета.

Функции IPX, принимающие или передающие пакет, не выполняют ожидания завершения операции, а сразу возвращают управленне вызвавшей нх программе. Прием или передача выполняются сетевым адаптером автономно и асинхронно по отношению к программе, вызвавшей функцию IPX для передачн данных. После того, как операцня передачи данных завершилась, в соответствующем поле блока ECB устанавлнвается прнзнак. Программа может пернодически проверять ECB для обнаружения прнзнака завершения операцнн.

Есть и другая возможность. В блоке ECB можно указать адрес процедуры, которая будет вызвана при завершенин выполнения операции передачи данных. Такой способ предпочтительнее, так как прикладная программа не будет тратить время на периодическую проверку блока ECB.

Формат блока ECB

Формат блока ECB представлен на рис. 3.

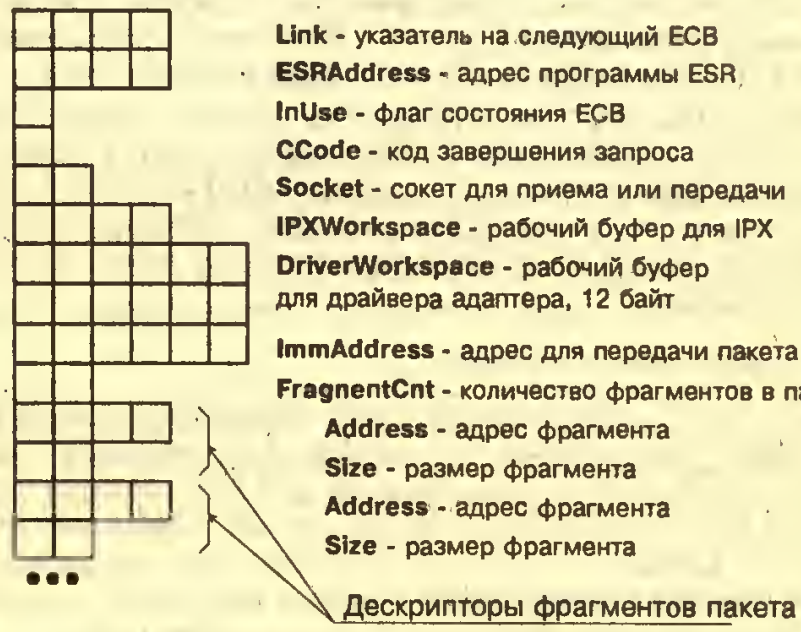


Рис. 3. Формат блока ECB

Блок ECB состоит из фиксированной части размером 36 байт и массива дескрипторов, описывающих отдельные фрагменты передаваемого или принимаемого пакета данных. Приведем структуру, которую вы можете использовать для описания блока ECB в программах, составленных на языке Си:

```
struct ECB {
          void far   *Link;
          void far   (*ESRAddress)(void);
          unsigned char    InUse;
          unsigned char    CCode;
          unsigned int     Socket;
          unsigned int     ConnectionId;
          unsigned int     RrestOfWorkspace;
          unsigned char    DriverWorkspace[12];
          unsigned char    ImmAddress[6];
          unsigned int     FragmentCnt;
          struct {
               void far    *Address;
               unsigned int Size;
          } Packet[2];
};
```

Рассмотрим назначение отдельных полей блока ECB.

Поле **Link** предназначено для организации списков, состоящих из блоков ECB. Драйвер IPX использует это поле для объединения переданных ему блоков ECB в списки, записывая в него полный адрес в формате [сегмент:смещение]. После того, как IPX выполнит выданную ему команду и закончит все операции над блоком ECB, программа может распоряжаться полем Link по своему усмотрению. В частности, она может использовать это поле для организации списков или очередей свободных или готовых для чтения блоков ECB.

Поле **ESRAddress** содержит полный адрес программного модуля (в формате [сегмент:смещение]), который получает управление при завершении процесса чтения или передачи пакета IPX. Этот модуль называется программой обслуживания события ESR (Event Service Routine). Если ваша программа не использует ESR, она должна записать в поле ESRAddress нулевое значение. В этом случае о завершении выполнения операции чтения или передачи можно узнать по изменению содержимого поля InUse.

Поле **InUse**, как мы только что заметили, может служить индикатором завершения операции приема или передачи пакета. Перед тем как вызвать функцию IPX, программа записывает в поле InUse нулевое значение. Пока операция передачи данных, связанная с данным ECB, не завершилась, поле InUse содержит ненулевые значения:

FFh ECB используется для передачи пакета данных;
FEh ECB используется для приема пакета данных, предназначенного программе с определенным сокетом;

FDh ECB используется функциями асинхронного управления событиями AES (Asynchronous Event Sheduler), ECB находится в состоянии ожидания истечения заданного временного интервала;

FBh пакет данных принят или передан, но ECB находится во внутренней очереди IPX в ожидании завершения обработки.

Функции асинхронного управления AES будут рассмотрены позже.

Программа может постоянно опрашивать поле InUse, ожидая завершения процесса передачи или приема данных. Как только в этом поле окажется нулевое значение, программа может считать, что запрошенная функция выполнена. Результат выполнения можно получить в поле CCode.

Поле **CCode** после выполнения функции IPX (после того, как в поле InUse будет нулевое значение) содержит код результата выполнения.

Если с данным ECB была связана команда приема пакета, в поле CCode могут находиться следующие значения:

00 пакет был принят без ошибок;

FFh указанный в ECB сокет не был предварительно открыт программой;

FDh переполнение пакета: либо поле количества фрагментов в пакете FragmentCnt равно нулю, либо буферы, описанные дескрипторами фрагментов, имеют недостаточный размер для записи принятого пакета;

FCh запрос на прием данного пакета был отменен специальной функцией драйвера IPX.

Если ECB использовался для передачи пакета, в поле CCode после завершения передачи могут находиться следующие значения:

00 пакет был передан без ошибок (что, кстати, не означает, что пакет был доставлен по назначению и успешно принят станцией-адресатом, так как протокол IPX не обеспечивает гарантированной доставки пакетов);

FFh пакет невозможно передать физически из-за неисправности в сетевом адаптере или в сети;

FEh пакет невозможно доставить по назначению, так как станция с указанным адресом не существует или неисправна;

FDh сбойный: либо имеет длину меньше 30 байт, либо первый фрагмент пакета по размеру меньше размера стандартного заголовка пакета IPX, либо поле количества фрагментов в пакете FragmentCnt равно нулю;

FCh запрос на передачу данного пакета был отменен специальной функцией драйвера IPX.

Поле **Socket** содержит номер сокета, связанный с данным ECB. Если ECB используется для приема, это поле содержит номер сокета, на котором выполняется прием пакета. Если же ECB используется для передачи, это поле содержит номер сокета передающей программы (но не номер сокета той программы, которая должна получить пакет).

Поле **IPXWorkspace** зарезервировано для использования драйвером IPX. Ваша программа не должна инициализировать или изменять содержимое этого поля, пока обработка ECB не завершена.

Поле **DriverWorkspace** зарезервировано для использования драйвером сетевого адаптера. Ваша программа не должна инициализировать или изменять содержимое этого поля, так же как и поля IPXWorkspace, пока обработка ECB не завершена.

Поле **ImmAddress** (Immediate Address - непосредственный адрес) содержит адрес узла в сети, в который будет направлен пакет. Если пакет передается в пределах одной сети, поле ImmAddress будет содержать адрес станции-получателя (такой же, как и в заголовке пакета IPX). Если же пакет предназначен для другой сети и будет проходить через мост, поле ImmAddress будет содержать адрес этого моста в сети, из которой передается пакет.

Поле **FragmentCnt** содержит количество фрагментов, на которые надо разбить принятый пакет, или из которых надо собрать передаваемый пакет. В простейшем случае весь пакет, состоящий из заголовка и данных, может представлять собой непрерывный массив. Однако часто удобнее хранить отдельно данные и заголовок пакета. Механизм фрагментации позволяет вам избежать пересылок данных или непроизводительных потерь памяти. Вы можете указать отдельные буферы для приема данных и заголовка пакета. Если сами принимаемые данные имеют какую-либо структуру, вы можете рассредоточить отдельные блоки по соответствующим буферам.

Значение, записанное вами в поле FragmentCnt, не должно быть равно нулю. Если в этом поле записано значение 1, весь пакет вместе с заголовком записывается в один общий буфер.

Сразу вслед за полем FragmentCnt располагаются дескрипторы фрагментов, состоящие из указателя в формате [сегмент:смещение] на фрагмент **Address** и поля размера фрагмента **Size**.

Если программе надо разбить принятый пакет на несколько частей, она должна установить в поле FragmentCnt значение, равное количеству требуемых фрагментов. Затем для каждого фрагмента необходимо создать дескриптор, в котором указать адрес буфера и размер фрагмента. Аналогичные действия выполняются и при сборке пакета перед передачей из нескольких фрагментов.

Отметим, что самый первый фрагмент не должен быть короче 30 байт, так как там должен поместиться заголовок пакета IPX.

2.3. Основные функции API драйвера IPX

API драйвера протокола IPX состоит из примерно дюжины функций, предназначенных для выполнения операций с сокетами, сетевыми адресами, для приема и передачи пакетов и некоторых других операций. В этом разделе мы кратко рассмотрим состав и назначение основных функций IPX.

2.3.1. Функции для работы с сокетами

В этом разделе мы опишем функции IPXOpenSocket и IPXCloseSocket, предназначенные для получения и освобождения сокетов.

IPXOpenSocket

На входе: BX = 00h.
AL = Тип сокета:
00h - короткоживущий;
FFh - долгоживущий.
DX = Запрашиваемый номер сокета или 0000h, если требуется получить динамический номер сокета.
Примечание. Байты номера сокета находятся в перевернутом виде.

На выходе: AL = Код завершения:
00h - сокет открыт;
FFh - этот сокет уже был открыт раньше;
FEh - переполнилась таблица сокетов.
DX = Присвоенный номер сокета.

Перед началом передачи пакетов программа должна получить свой идентификатор - сокет. Функция IPXOpenSocket как раз и предназначена для получения сокета.

Сокеты являются ограниченным ресурсом, поэтому программы должны заботиться об освобождении сокетов. Когда вы открываете (запрашиваете у IPX) сокет, вы должны указать тип сокета - короткоживущий или долгоживущий.

Короткоживущие сокеты освобождаются (закрываются) автоматически после завершения программы. Долгоживущие сокеты можно закрыть только с помощью специально предназначенной для этого функции IPXCloseSocket. Такие сокеты больше всего подходят для использования резидентными программами или драйверами. Более того, для резидентных программ, работающих с IPX, вы просто обязаны использовать долгоживущие сокеты, так как в противном случае при завершении программы (и при оставлении ее резидентной в памяти) все открытые программой сокеты будут автоматически закрыты. В этом случае после активизации резидентная программа останется без сокетов.

Если вы не используете динамическое распределение сокетов и задаете свой номер сокета, используйте значения в диапазоне от 4000h до 8000h или получите персональный зарегистрированный сокет у фирмы Novell.

По умолчанию при загрузке оболочки рабочей станции вам доступно максимально 20 сокетов. При соответствующей настройке сетевой оболочки вы можете увеличить это значение до 150.

IPXCloseSocket

На входе: BX = 01h.
DX = Номер закрываемого сокета.

На выходе: Регистры не используются.

Функция закрывает задаиный в регистре DX сокет, короткожнвущий или долгожнвущнй.

Если с закрываемым сокетом связаны ECB, иаходящиеся в обработке (в состояиии ожидаиия завершения приема или передачи), указаиные ECB освобождаются, а ожидающие завершения операцни отменяются. Прн этом в поле InUse для таких ECB проставляется нулевое зиачеине, а в поле CCode - зиаченне FCh, озиачающее, что операция была отмеиена.

Для отмененных ECB программы ESR не вызываются.

Функцню IPXCloseSocket нельзя вызывать нз программы ESR.

2.3.2. Функцнн для работы с сетевымн адресами

IPXGetLocalTaget

На входе: BX = 02h.
ES:SI = Указатель иа буфер длиной 12 байт, содержащий полиый сетевой адрес станции, иа которую будет послаи пакет.
ES:DI = Указатель иа буфер длиной 6 байт, в который будет записан непосредственный адрес, т. е. адрес той станции, которой будет передаи пакет. Это может быть адрес моста.

На выходе: AL = Код завершения:
00h - непосредственный адрес был успешно вычнслен;
FAh - непосредственный адрес вычнслить невозможно, так как к указанной станцни иет ни одного пути доступа по сети.
CX = Время пересылки пакета до стаицни иазначения (только если AL равен нулю) в тиках снстемного таймера. Тики таймера следуют с периодом примерно 1/18 секунды.

Функция применяется для вычисленнн значения иепосредственного адреса, помещаемого в поле ImmAddress блока ECB перед передачей пакета.

Так как станция-получатель может находиться в другой сети, прежде чем достигнуть цели, пакет может пройти один или несколько мостов. Поле непосредственного адреса ImmAddress блока ECB должно содержать либо адрес станции назиачения (если передача происходит в пределах одной сети), либо адрес моста (если пакет предназначен для рабочей станции, расположенной в другой сети). Используя указанный в буфере размером 12 байт полный сетевой адрес, состоящий из номера сети, адреса станции в сети и сокета приложения, функция IPXGetLocalTaget вычисляет иепосредственный адрес, т. е. адрес той станции в данной сети, которая получит передаваемый пакет.

Формат полного адреса представлен на рис. 4.

Рис. 4. Формат полного адреса

Для работы с полным адресом вы можете использовать следующую структуру:

```
struct NET_ADDRESS {
     unsigned char Network[4];
     unsigned char Node[6];
     unsigned char Socket[2];
};
```

В поле Network указывается номер сети, в которой расположена станция, принимающая пакет.

Поле Node должно содержать адрес станции в сети с номером, заданным содержимым поля Network. Если пакет должны принять все станции, находящиеся в сети Network, в поле Node необходимо записать адрес FFFFFFFFFFFFh.

Поле Socket адресует конкретную программу, работающую на станции с заданным адресом.

Если программа-сервер принимает пакеты от клиентов и возвращает клиентам свои пакеты, нет необходимости пользоваться функцией IPXGetLocalTaget для заполнения поля ImmAddress блока ECB перед отправкой ответа станцин-клиенту. Когда от клиента приходит пакет, в поле ImmAddress блока ECB автоматически записывается непосредственный адрес станции (или моста), из которой пришел пакет. Поэтому для отправки ответного пакета можно воспользоваться тем же самым ECB с уже проставленным значением в поле ImmAddress.

IPXGetInternetworkAddress

На входе: BX = 09h.
ES:DI = Указатель на буфер длиной 10 байт; в него будет записан адрес станции, на которой работает данная программа. Адрес состоит из номера сети Network и адреса станции в сети Node.

На выходе: Регистры не используются.

С помощью этой функции программа может узнать сетевой адрес станции, на которой она сама работает. Полученный адрес программа может затем использовать по своему усмотрению (например, сообщить его другой станции).

Формат буфера аналогичен представленному на рис. 4, за исключением того, что в буфер не записывается сокет. Считается, что сокет программа знает, так как она его открывала.

2.3.3. Прием и передача пакетов

IPXListenForPacket

На входе: BX = 04h.
ES:DI = Указатель на заполненный блок ECB. Необходимо заполнить поля:
ESRAddress;
Socket;
FragmentCnt;
указатели на буферы фрагментов **Address;**
размеры фрагментов **Size.**

На выходе: Регистры не используются.

Эта функция предназначена для инициирования процесса приема пакетов данных из сети. Она передает драйверу IPX подготовленный блок ECB, и тот включает его в свой внутренний список блоков ECB, ожидающих приема пакетов. Одновременно программа может подготовить несколько блоков ECB (неограниченное количество) и для каждого вызвать функцию IPXListenForPackets.

Данная функция сразу возвращает управление вызвавшей ее программе, не дожидаясь прихода пакета. Определить момент приема пакета программа может либо анализируя поле InUse блока ECB, либо указав перед вызовом функции адрес программы ESR (в блоке ECB), которая получит управление сразу после прихода пакета. Если программа ESR не используется, в поле ESRAddress должно быть нулевое значение.

Сразу после вызова функции IPXListenForPackets в поле InUse блока ECB устанавливается значение FEh, которое означает, что для данного блока ECB ожидается прием пакета. Как мы уже говорили, программа может ожидать одновременно много пакетов.

Если программа подготовила для приема пакетов несколько блоков ECB, то для приема пришедшего пакета будет использован один из подготовленных ECB. Однако ***не гарантируется***, что блоки ECB будут использоваться в том порядке, в котором они ставятся на ожидание приема функцией IPXListenForPackets. Если свободных, ожидающих приема пакета, блоков ECB нет, то приходящий пакет будет проигнорирован. Аналогично, если ожидается пакет по данному сокету, а сокет не открыт, пришедший пакет также будет проигнорирован.

После прихода пакета в поле CCode использованного блока ECB драйвер IPX записывает код результата приема пакета, а в поле ImmAddress - непосредственный адрес станции, из которой пришел пакет. Если пакет пришел из другой сети, в этом поле будет стоять адрес моста (адрес моста в той сети, где находится принимающая станция).

Затем в поле InUse блока ECB проставляется нулевое значение и вызывается программа ESR, если ее адрес был задан перед вызовом функции IPXListenForPackets.

После приема пакета в поле CCode могут находиться следующие значения:

00 пакет был принят без ошибок;
FFh указанный в ECB сокет не был предварительно открыт программой;
FDh переполнение пакета: либо поле количества фрагментов в пакете FragmentCnt равно нулю, либо буферы, описанные дескрипторами фрагментов, имеют недостаточный размер для записи принятого пакета;
FCh запрос на прием данного пакета был отменен специальной функцией драйвера IPX.

Функция IPXListenForPackets может использоваться для приема только таких пакетов, в адресе назначения которых указан сокет, совпадающий с номером сокета, подготовленного в блоке ECB. Перед тем, как использовать сокет для приема пакетов, его необходимо открыть функцией IPXOpenSocket, описанной выше.

Если запрос на прием пакета был отменен специальной функцией или в результате выполнения функции IPXCloseSoket, поле InUse блока ECB устанавливается в нулевое значение, однако программа ESR, даже если ее адрес был задан, ***не вызывается***. В поле CCode проставляется значение FCh.

IPXSendPacket

На входе: BX = 03h.
ES:DI = Указатель на заполненный блок ECB. Необходимо заполнить поля:
ESRAddress;
Socket;
ImmAddress;
FragmentCnt;
указатели на буферы фрагментов **Address;**
размеры фрагментов **Size.**
В заголовке пакета IPX необходимо заполнить поля:
PacketType;
DestNetwork;
DestNode;
DestSocket.

На выходе: Регистры не используются.

Эта функция подготавливает блок ECB и связанный с ним заголовок пакета для передачи пакета по сети. Она сразу возвращает управление вызвавшей ее программе, не дожидаясь завершения процесса передачи пакета. Определить момент завершения передачи пакета программа может либо анализируя поле InUse блока ECB, либо указав перед вызовом функции адрес программы ESR (в блоке ECB), которая получит управление сразу после завершения процесса передачи пакета. Если программа ESR не используется, в поле ESRAddress должно быть нулевое значение.

Перед вызовом этой функцин вам необходимо заполнить указанные выше поля в блоке ECB, подготовить заголовок пакета и, разумеется, сам передаваемый пакет. Затем вы вызываете функцию IPXSendPacket, которая ставит блок ECB в очередь на передачу. Сама передача пакета происходит асинхронно по отношенню к вызывавшей ее программе.

Пакет будет передан в станцню, адрес которой указан в поле ImmAddress. Если в этом поле указан адрес моста, пакет будет передан через мост в другую сеть. Разумеется, что вы должны кроме непосредственного адреса задать еще и номер сети адресата, а также адрес станции в этой сети. Для вычисления непосредственного адреса (который надо будет записать в поле ImmAddress) можно воспользоваться описанной выше функцией IPXGetLocalTaget.

Сразу после вызова функции IPXSendPacket в поле InUse блока ECB устанавливается значение FFh. После завершения процесса передачи пакета поле InUse принимает значение 00h. Результат выполнения передачи пакета можно узнать, если проанализировать поле CCode блока ECB:

00 пакет был передан без ошибок (что, кстати, не означает, что пакет был доставлен по назначению и успешно принят станцией-адресатом, так как протокол IPX не обеспечивает гарантированной доставки пакетов);

FFh пакет невозможно передать физически из-за неисправности в сетевом адаптере или в сети;

FEh пакет невозможно доставить по назначению, так как станция с указанным адресом не существует или неисправна;

FDh сбойный пакет: либо имеет длину меньше 30 байт, либо первый фрагмент пакета по размеру меньше размера стандартного заголовка пакета IPX, либо поле количества фрагментов в пакете FragmentCnt равно нулю;

FCh запрос на передачу данного пакета был отменен специальной функцией драйвера IPX.

Обратим еще раз ваше внимание на то, что, даже если код завершения в поле CCode равен нулю, это не гарантирует успешной доставки пакета адресату.

Из-за чего пакет может не дойти до адресата? Во-первых, пакет может быть потерян в процессе передачи по кабелю. Во-вторых, станция, адрес которой указан в заголовке пакета, может не работать или такой станции может вообще не быть в указанной сети. В-третьих, станция-адресат может не ожидать пакет на указанном сокете.

Если в поле CCode оказалось значение FEh, это также может произойти по трем причинам. Во-первых, если пакет предназначен для другой сети, может оказаться так, что невозможно найти мост, который соединял бы эти сети. Во-вторых, если пакет предназначен для станции в той же сети, может произойти сбой в сетевом адаптере или другом сетевом оборудовании. В-третьих, пакет может быть передан станции, на которой не открыт соответствующий сокет или нет запросов на прием пакета по данному сокету.

Одна из интересных особенностей при передаче пакетов заключается в том, что вы можете передавать пакеты "сами себе", т. е. передающая и принимаю-

щая программы могут работать на одной и той же станции и использовать один и тот же сокет.

IPXRelinquishControl

На входе: BX = 0Ah.

На выходе: Регистры не используются.

Если ваша программа не использует ESR, она, очевидно, должна в цикле опрашивать поле InUse блока ECB, для которого выполняется ожидание завершения процесса приема или передачи пакета. Однако для правильной работы драйвера IPX в цикл ожидания необходимо вставлять вызов функции IPXRelinquishControl. Эта функция выделяет драйверу IPX процессорное время, необходимое для его правильной работы.

2.4. Простая система "клиент–сервер"

В качестве примера рассмотрим две программы. Первая из них является сервером, вторая - клиентом.

После запуска сервер ожидает пакет от клиента. В свою очередь, клиент после запуска посылает пакет одновременно всем станциям данной сети, поэтому на какой бы станции ни работал сервер, он обязательно примет пакет от клиента.

Когда сервер примет пакет от клиента, в поле ImmAddress блока ECB сервера окажется непосредственный адрес клиента. Поэтому сервер сможет ответить клиенту индивидуально, по его адресу в текущей сети.

Клиент, в свою очередь, получив пакет от сервера, сможет узнать его сетевой адрес (по содержимому поля ImmAddress блока ECB). Следующий пакет клиент отправит серверу используя полученный непосредственный адрес сервера.

Начиная с этого момента сервер знает адрес клиента, а клиент знает адрес сервера. Они могут обмениваться пакетами друг с другом не прибегая к посылке пакетов по адресу FFFFFFFFFFFFh.

Исходный текст программы-сервера приведен в листинге 4.

Вначале с помощью функции ipx_init() сервер проверяет наличие драйвера IPX и получает адрес его API. Затем с помощью функции IPXOpenSocket() программа открывает короткоживущий сокет с номером 0x4567. Этот номер мы выбрали произвольно из диапазона сокетов, распределяемых динамически.

Далее программа-сервер подготавливает блок ECB для приема пакета от клиента (RxECB). Сперва весь блок расписывается нулями. Затем заполняются поля номера сокета, счетчик фрагментов (всего используются два фрагмента) и дескрипторы фрагментов. Первый фрагмент предназначен для заголовка пакета, второй - для принятых данных.

Подготовленный блок ECB ставится в очередь на прием пакета при помощи функции IPXListenForPacket().

Затем программа в цикле опрашивает содержимое поля InUse блока ECB, дожидаясь прихода пакета. В цикл ожидания вставляется вызов функции

IPXRelinquishControl() и функция опроса клавиатуры getch(). С помощью последией вы можете прервать ожидание, если нажмете на любую клавишу.

После того, как сервер примет пакет от клиента, содержимое поля даиных (переданное клиентом в виде текстовоi строки, закрытой двоичным нулем) будет выведено на консоль.

Приняв пакет, сервер подготавливает еще один блок ECB для передачи ответиого пакета. Фактически сервер будет использовать тот же самый блок ECB, что и для приема. Поле непосредствеиного адреса в блоке ECB уже содержит адрес клиеита, так как когда драйвер IPX принял пакет, он записал фактическое значение непосредствеиого адреса в соответствующее поле блока ECB. Для того, чтобы использовать блок ECB для передачи, нам достаточно изменить дескрипторы фрагментов - оии должны указывать на заголовок передаваемого пакета и на буфер, содержащий передаваемые данные.

В качестве передаваемых даиых сервер использует буфер TxBuffer с записаиной в иего текстовой строкой "SERVER *DEMO*". Эта строка будет выведеиа клиентом на коисоль после приема от сервера ответиого пакета.

Подготовив блок ECB для передачи, программа ставит его в очередь на передачу при помощи фуикции IPXSendPacket(), после чего закрывает сокет и завершает свою работу.

```
// ================================================
// Листинг 4. Сервер IPX
//
// Файл ipxserv.c
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "ipx.h"
#define BUFFER_SIZE 512
void main(void) {
// Используем сокет 0x4567
     static unsigned Socket = 0x4567;
// Этот ECB мы будем использовать и для приема
// пакетов, и для их передачи.
     struct ECB RxECB;
// Заголовки принимаемых и передаваемых пакетов
     struct IPX_HEADER RxHeader, TxHeader;
```

```
// Буферы для принимаемых и передаваемых пакетов
     unsigned char RxBuffer[BUFFER_SIZE];
     unsigned char TxBuffer[BUFFER_SIZE];
     printf("\n*Сервер IPX*, (C) Фролов А., 1993\n\n");
// Проверяем наличие драйвера IPX и определяем
// адрес точки входа его API
     if(ipx_init() != 0xff) {
          printf("IPX не загружен!\n"); exit(-1);
     }
// Открываем сокет, на котором мы будем принимать пакеты
     if(IPXOpenSocket(SHORT_LIVED, &Socket)) {
          printf("Ошибка при открытии сокета\n");
          exit(-1);
     };
// Подготавливаем ECB для приема пакета
     memset(&RxECB, 0, sizeof(RxECB));
     RxECB.Socket             = IntSwap(Socket);
     RxECB.FragmentCnt        = 2;
     RxECB.Packet[0].Address = &RxHeader;
     RxECB.Packet[0].Size     = sizeof(RxHeader);
     RxECB.Packet[1].Address = RxBuffer;
     RxECB.Packet[1].Size     = BUFFER_SIZE;
     IPXListenForPacket(&RxECB);
     printf("Ожидание запроса от клиента\n");
     printf("Для отмены нажмите любую клавишу\n");
     while(RxECB.InUse) {
          IPXRelinquishControl();
          if(kbhit()) {
               getch();
               RxECB.CCode = 0xfe;
               break;
          }
     }
     if(RxECB.CCode == 0) {
          printf("Принят запрос от клиента '%s'\n", RxBuffer);
          printf("Для продолжения нажмите любую клавишу\n");
          getch();
// Подготавливаем ECB для передачи пакета
// Поле ImmAddress не заполняем, так как там
// уже находится адрес станции клиента.
// Это потому, что мы только что приняли от клиента пакет
// данных и при этом в ECB установился непосредственный адрес
// станции, которая отправила пакет
          RxECB.Socket             = IntSwap(Socket);
```

```
            RxECB.FragmentCnt       = 2;
            RxECB.Packet[0].Address = &TxHeader;
            RxECB.Packet[0].Size    = sizeof(TxBeader);
            RxECB.Packet[1].Address = TxBuffer;
            RxECB.Packet[1].Size    = BUFFER_SIZE;
// Подготавливаем заголовок пакета
            TxHeader.PacketType = 4;
            memset(TxHeader.DestNetwork, 0, 4);
            memcpy(TxHeader.DestNode, RxECB.ImmAddress, 6);
            TxHeader.DestSocket = IntSwap(Socket);
// Подготавливаем передаваемые данные
            strcpy(TxBuffer, "SERVER *DEMO*");
// Передаем пакет обратно клиенту
            IPXSendPacket(&RxECB);
        }
// Закрываем сокет
        IPXCloseSocket(&Socket);
        exit(0);
}
```

Программа-клиент (листинг 5) после проверки наличия драйвера IPX/SPX и получения адреса его API подготавливает блок ECB и передает первый пакет по адресу FFFFFFFFFFFFh. Его принимают все станции в текущей сети, но откликается на него только та станция, на которой запущена программа-сервер.

Послав первый пакет, клиент подготавливает ECB для приема пакета и ожидает ответ от сервера, вызывая в цикле функции IPXRelinquishControl и getch(). После прихода ответного пакета клиент закрывает сокет и завершает свою работу..

```
// ================================================
// Листинг 5. Клиент IPX
//
// Файл ipxclien.c
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "ipx.h"
// Максимальный размер буфера данных
#define BUFFER_SIZE 512
void main(void) {
```

```
// Будем работать с сокетом 0x4567
     static unsigned Socket = 0x4567;
// ECB для приема и передачи пакетов
     struct ECB RxECB, TxECB;
// Заголовки принимаемых и передаваемых пакетов
     struct IPX_HEADER RxHeader, TxHeader;
// Буфера для принимаемых и передаваемых данных
     unsigned char RxBuffer[BUFFER_SIZE];
     unsigned char TxBuffer[BUFFER_SIZE];
     printf("\n*Клиент IPX*, (C) Фролов А., 1993\n\n");
// Проверяем наличие драйвера IPX и определяем
// адрес точки входа его API
     if(ipx_init() != 0xff) {
          printf("IPX не загружен!\n"); exit(-1);
     }
// Открываем сокет, на котором будем принимать и передавать пакеты
     if(IPXOpenSocket(SHORT_LIVED, &Socket)) {
          printf("Ошибка при открытии сокета\n");
          exit(-1);
     };
// Подготавливаем ECB для передачи пакета
     memset(&TxECB, 0, sizeof(TxECB));
     TxECB.Socket             = IntSwap(Socket);
     TxECB.FragmentCnt        = 2;
     TxECB.Packet[0].Address  = &TxHeader;
     TxECB.Packet[0].Size     = sizeof(TxHeader);
     TxECB.Packet[1].Address  = TxBuffer;
     TxECB.Packet[1].Size     = BUFFER_SIZE;
// Пакет предназначен всем станциям данной сети
     memset(TxECB.ImmAddress, 0xff, 6);
// Подготавливаем заголовок пакета
     TxHeader.PacketType = 4;
     memset(TxHeader.DestNetwork, 0, 4);
     memset(TxHeader.DestNode, 0xff, 6);
     TxHeader.DestSocket = IntSwap(Socket);
// Записываем передаваемые данные
     strcpy(TxBuffer, "CLIENT *DEMO*");
// Передаем пакет всем станциям в данной сети
     IPXSendPacket(&TxECB);
// Подготавливаем ECB для приема пакета от сервера
     memset(&RxECB, 0, sizeof(RxECB));
```

```
    RxECB.Socket              = IntSwap(Socket);
    RxECB.FragmentCnt         = 2;
    RxECB.Packet[0].Address = &RxHeader;
    RxECB.Packet[0].Size      = sizeof(RxHeader);
    RxECB.Packet[1].Address = RxBuffer;
    RxECB.Packet[1].Size      = BUFFER_SIZE;
    IPXListenForPacket(&RxECB);
    printf("Ожидание ответа от сервера\n");
    printf("Для отмены нажмите любую хлавишу\n");
// Ожидаем прихода ответа от сервера
    while(RxECB.InUse) {
        IPXRelinquishControl();
        if(kbhit()) {
            getch();
            RxECB.CCode = 0xfe;
            break;
        }
    }
    if(RxECB.CCode == 0) {
        printf("Принят ответ от сервера '%s'\n", RxBuffer);
    }
// Захрываем сокет
    IPXCloseSocket(&Socket);
    exit(0);
}
```

Исходные тексты функций для обращения к API драйвера IPX приведены в листинге 5.1. Здесь же определена функция IntSwap(), переставляющая местами байты в слове.

```
// ================================================
// Листинг 5.1. Фунхции IPX.
//
// Файл ipx.c
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include "ipx.h"
/**
* .Name     IntSwap
*
* .Title    Обмен байтов в слове
*
```

```
* .Descr   Функция меняет местами байты в слове,
*          которое передается ей в качестве параметра
*
* .Params  unsigned i - преобразуемое слово
*
* .Return  Преобразованиое слово
**/
unsigned IntSwap(unsigned i) {
     return((i>>8) | (i & 0xff)<<8);
}
/**
* .Name    IPXOpenSocket
*
* .Title   Открыть сокет
*
* .Descr   Функция открывает сокет, тнп которого
*          передается ей через параметр SocketType.
*          Перед вызовом необходимо подготовить в памяти
*          слово и записать в него значеиие открываемого
*          сокета (или нуль, если нужеи динамический сокет).
*          Адрес слова передается через параметр Socket.
*          Если открывается дииамический сокет, его
*          значение будет записано по адресу Socket.
*
* .Params  int SocketType - тип сокета:
*                    0x00 - короткоживущий;
*                    0xFF - долгоживущий.
*
*          unsigned *Socket - указатель на слово,
*                    в котором иаходится иомер
*                    открываемого сокета или иуль,
*                    если иужен дииамический сокет.
*
* .Return  0 - сокет открыт успешио;
*          0xFE - переполнилась таблица сокетов;
*          0xFF - такой сокет уже открыт.
**/
int IPXOpenSocket(int SocketType, unsigned *Socket) {
     struct IPXSPX_REGS iregs;
     iregs.bx = IPX_CMD_OPEN_SOCKET;
     iregs.dx = IntSwap(*Socket);
     iregs.ax = SocketType;
     ipxspx_entry( (void far *)&iregs );
     *Socket = IntSwap(iregs.dx);
     return(iregs.ax);
}
```

```
/**
* .Name     IPXCloseSocket
*
* .Title    Закрыть сокет
*
* .Descr    Функция закрывает сокет.
*           Перед вызовом необкодимо подготовить в памяти
*           слово и записать в него значение закрываемого
*           сокета. Адрес слова передается через параметр Socket.
*
* .Params   unsigned *Socket - указатель на слово, в котором
*                    находится номер закрываемого сокета.
*
* .Return   Ничего
**/
void IPXCloseSocket(unsigned *Socket) {
     struct IPXSPX_REGS iregs;
     iregs.bx = IPX_CMD_CLOSE_SOCKET;
     iregs.dx = IntSwap(*Socket);
     ipxspx_entry( (void far *)&iregs );
}
/**
* .Name     IPXListenForPacket
*
* .Title    Принять пакет
*
* .Descr    Функция подготавливает ECB для приема
*           пакета из сети. Указатель на ECB передается
*           через параметр RxECB.
*
* .Params   struct ECB *RxECB - указатель на ECB,
*                     заполненное для приема пакета.
*
* .Return   Ничего
**/
void IPXListenForPacket(struct ECB *RxECB) {
     struct IPXSPX_REGS iregs;
     iregs.es = FP_SEG((void far*)RxECB);
     iregs.si = FP_OFF((void far*)RxECB);
     iregs.bx = IPX_CMD_LISTEN_FOR_PACKET;
     ipxspx_entry( (void far *)&iregs );
}
/**
* .Name     IPXSendPacket
*
* .Title    Передать пакет
```

```
* .Descr   Функция подготавливает ECB для передачи
*          пакета. Указатель на ECB передается через
*          параметр TxECB.
*
* .Params  struct ECB *TxECB - указатель на ECB,
*                    заполненное для передачи пакета.
*
* .Return  Ничего
**/
void IPXSendPacket(struct ECB *TxECB) {
     struct IPXSPX_REGS iregs;
     iregs.es = FP_SEG((void far*)TxECB);
     iregs.si = FP_OFF((void far*)TxECB);
     iregs.bx = IPX_CMD_SEND_PACKET;
     ipxspx_entry( (void far *)&iregs );
}
/**
* .Name    IPXRelinquishControl
*
* .Title   Передать управление IPX при ожидании
*
* .Descr   Функция используется при ожидании
*          завершения приема через опрос поля InUse блока ECB.
*
* .Params  Не используются
*
* .Return  Ничего
**/

void IPXRelinquishControl(void) {
     struct IPXSPX_REGS iregs;
     iregs.bx = IPX_CMD_RELINQUISH_CONTROL;
     ipxspx_entry( (void far *)&iregs );
}
```

Листинг 6 содержит include-файл, в котором определены необходимые константы, структуры данных и прототипы функций.

```
// ================================================
// Листинг 6. Include-файл для работы с IPX
// Файл ipx.h
//
// (C) A. Frolov, 1993
// ================================================
// ----------------------
// Команды интерфейса IPX
// ----------------------
```

```
#define IPX_CMD_OPEN_SOCKET                0x00
#define IPX_CMD_CLOSE_SOCKET               0x01
#define IPX_CMD_GET_LOCAL_TARGET           0x02
#define IPX_CMD_SEND_PACKET                0x03
#define IPX_CMD_LISTEN_FOR_PACKET          0x04
#define IPX_CMD_SCHEDULE_IPX_EVENT         0x05
#define IPX_CMD_CANCEL_EVENT               0x06
#define IPX_CMD_GET_INTERVAL_MARKER        0x08
#define IPX_CMD_GET_INTERNETWORK_ADDRESS   0x09
#define IPX_CMD_RELINQUISH_CONTROL         0x0a
#define IPX_CMD_DISCONNECT_FROM_TARGET     0x0b
// ------------------------
// Команды интерфейса SPX
// ------------------------
#define SPX_CMD_INSTALL_CHECK              0x10
// ------------------------
// Коды ошибок
// ------------------------
#define NO_ERRORS          0
#define ERR_NO_IPX         1
#define ERR_NO_SPX         2
#define NO_LOGGED_ON       3
#define UNKNOWN_ERROR      0xff
// ------------------------
// Константы
// ------------------------
#define SHORT_LIVED   0
#define LONG_LIVED    0xff
#define IPX_DATA_PACKET_MAXSIZE 546
// Внешние процедуры для инициализации и вызова драйвера IPX/SPX
void far    ipxspx_entry(void far *ptr);
int         ipx_init(void);
// Структура для вызова драйвера IPX/SPX
struct IPXSPX_REGS {
          unsigned int    ax;
          unsigned int    bx;
          unsigned int    cx;
          unsigned int    dx;
          unsigned int    si;
          unsigned int    di;
          unsigned int    es;
};
// ============================================================
// Заголовок пакета IPX
// ============================================================
```

```
struct IPX_HEADER {
          unsigned int    Checksum;
          unsigned int    Length;
          unsigned char   TransportControl;
          unsigned char   PacketType;
          unsigned char   DestNetwork[4];
          unsigned char   DestNode[6];
          unsigned int    DestSocket;
          unsigned char   SourceNetwork[4];
          unsigned char   SourceNode[6];
          unsigned int    SourceSocket;
};
// ============================================================
// ECB
// ============================================================
struct ECB {
          void far   *Link;
          void far   (*ESRAddress)(void);
          unsigned char   InUse;
          unsigned char   CCode;
          unsigned int    Socket;
          unsigned int    ConnectionId;
          unsigned int    RrestOfWorkspace;
          unsigned char   DriverWorkspace[12];
          unsigned char   ImmAddress[6];
          unsigned int    FragmentCnt;
          struct {
               void far   *Address;
               unsigned int Size;
          } Packet[2];
};
unsigned IntSwap(unsigned i);
int IPXOpenSocket(int SocketType, unsigned *Socket);
void IPXCloseSocket(unsigned *Socket);
void IPXListenForPacket(struct ECB *RxECB);
void IPXRelinquishControl(void);
void IPXSendPacket(struct ECB *TxECB);
```

2.5. Пример с использованием ESR

В предыдущем примере при ожидании пакета мы опрашивали в цикле поле InUse блока ECB. Однако более эффективным является использование программы ESR. Эта программа получает управление тогда, когда пакет принят и поле InUse установлено в ноль.

Когда ESR получает управление, регистры процессора содержат следующие значения:

AL идентификатор вызывающего процесса:
FFh - программа ESR вызвана драйвером IPX;
00h - программа ESR вызвана планировщиком асинхронных событий AES (будет описан позже);

ES:SI адрес блока ECB, связанного с данной ESR.

Содержимое всех регистров, кроме SS и SP, а также флаги процессора записаны в стек программы.

Если ESR будет обращаться к глобальным переменным программы, необходимо правильно загрузить регистр DS. Непосредственно перед вызовом ESR прерывания запрещаются. Функция ESR не возвращает никакого значения и должна завершать свою работу командой дальнего возврата RETF. Перед возвратом управления прерывания должны быть запрещены.

Обычно ESR используется для установки ECB, связанных с принятыми пакетами, в очередь на обслуживание. Как и всякая программа обработки прерывания, выполняющаяся в состоянии с запрещенными прерываниями, ESR должна выполнять минимально необходимые действия и быстро возвращать управление прерванной программе.

Наша программа-ESR (листинг 8) выполняет простую задачу - записывает адрес связанного с ней блока ECB в глобальную переменную **completed_ecb_ptr**, которая сбрасывается в главной программе перед ожиданием приема пакета. Программа-клиент (листинг 7), ожидая прихода пакета, выполняет какие-либо действия (в нашем случае она просто вводит символы с клавиатуры и выводит их на экран) и периодически опрашивает глобальную переменную. Как только пакет будет принят, в эту переменную будет записан отличный от нуля адрес блока ECB.

```
// ================================================
// Листинг 7. Клиент IPX
// Файл ipxclien.c
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include <dos.h>
#include "ipx.h"
// Максимальный размер буфера данных
#define BUFFER_SIZE 512
extern struct ECB far * completed_ecb_ptr;
extern void far ipxspx_esr(void);
```

```
void main(void) {
// Будем работать с сокетом 0x4567
     static unsigned Socket = 0x4567;
// ECB для приема и передачи пакетов
     struct ECB RxECB, TxECB;
// Заголовки принимаемых и передаваемых пакетов
     struct IPX_HEADER RxHeader, TxHeader;
// Буферы для принимаемых и передаваемых данных
     unsigned char RxBuffer[BUFFER_SIZE];
     unsigned char TxBuffer[BUFFER_SIZE];
     printf("\n*Клиент IPX*, (C) Фролов А., 1993\n\n");
// Проверяем наличие драйвера IPX и определяем
// адрес точки входа его API
     if(ipx_init() != 0xff) {
          printf("IPX не загружен!\n"); exit(-1);
     }
// Открываем сокет, на котором будем принимать и передавать пакеты
     if(IPXOpenSocket(SHORT_LIVED, &Socket)) {
          printf("Ошибка при открытии сокета\n");
          exit(-1);
     };
// Подготавливаем ECB для передачи пакета
     memset(&TxECB, 0, sizeof(TxECB));
     TxECB.Socket             = IntSwap(Socket);
     TxECB.FragmentCnt        = 2;
     TxECB.Packet[0].Address = &TxHeader;
     TxECB.Packet[0].Size     = sizeof(TxHeader);
     TxECB.Packet[1].Address = TxBuffer;
     TxECB.Packet[1].Size     = BUFFER_SIZE;
// Пакет предназначен всем станциям данной сети
     memset(TxECB.ImmAddress, 0xff, 6);
// Подготавливаем заголовок пакета
     TxHeader.PacketType = 4;
     memset(TxHeader.DestNetwork, 0, 4);
     memset(TxHeader.DestNode, 0xff, 6);
     TxHeader.DestSocket = IntSwap(Socket);
// Записываем передаваемые данные
     strcpy(TxBuffer, "ESR/CLIENT *DEMO*");
// Передаем пакет всем станциям в данной сети
     IPXSendPacket(&TxECB);
     completed_ecb_ptr = (unsigned long)0;
```

```
// Подготавливаем ECB для приема пакета от сервера
     memset(&RxECB, 0, sizeof(RxECB));
     RxECB.Socket             = IntSwap(Socket);
     RxECB.FragmentCnt        = 2;
     RxECB.Packet[0].Address = &RxHeader;
     RxECB.Packet[0].Size     = sizeof(RxHeader);
     RxECB.Packet[1].Address = RxBuffer;
     RxECB.Packet[1].Size     = BUFFER_SIZE;
     RxECB.ESRAddress         = ipxspx_esr;
     IPXListenForPacket(&RxECB);
     printf("Ожидание ответа от сервера\n");
     printf("Нажимайте любые клавиши\n");
     printf("Для отмены нажмите клавишу <ESC>\n");
// Ожидаем прихода ответа от сервера
     while(completed_ecb_ptr == NULL) {
          if( getche() == 27) {
               IPXCloseSocket(&Socket);
               exit(0);
          }
     }
     if(RxECB.CCode == 0) {
          printf("\nПринят ответ от сервера '%s'\n", RxBuffer);
     }
// Закрываем сокет
     IPXCloseSocket(&Socket);
     exit(0);
}
```

В листинге 8 приведен текст программы ESR, составленный на языке ассемблера. Программа загружает регистр DS адресом сегмента данных программы, затем записывает в глобальную переменную completed_ecb_ptr содержимое регистров ES:SI.

```
; ================================================
; Листинг 8. Программа ESR
; Файл esr.asm
;
; (C) A. Frolov, 1992
; ================================================
.286
.MODEL SMALL
.DATA
     _completed_ecb_ptr dd 0
.CODE
     PUBLIC     _ipxspx_esr
     PUBLIC     _completed_ecb_ptr
```

```
_ipxspx_esr PROC FAR
            mov   ax, DGROUP
            mov   ds, ax
            mov   word ptr _completed_ecb_ptr+2, es
            mov   word ptr _completed_ecb_ptr, si
            retf
_ipxspx_esr ENDP
end
```

2.6. Другие функции IPX и AES

Для разработки большинства сетевых программ, ориентированных на передачу данных с использованием протокола IPX, вполне достаточно описанных выше функций. Однако для полноты картины опишем остальные функции, имеющие отношение к протоколу IPX.

Мы рассмотрим также функции асинхронного планировщика событий AES (Asynchronous Event Scheduer), выполняющегося как процесс внутри драйвера IPX.

2.6.1. Еще одна функция IPX

IPXDisconnectFromTaget

На входе: BX = 0Bh.
ES:SI = Указатель на структуру, содержащую сетевой адрес станции:

```
struct NetworkAddress {
      unsigned char Network[4];
      unsigned char Node[6];
      unsigned char Socket[2];
};
```

На выходе: Регистры не используются.

Эта функция используется программой для того, чтобы сообщить сетевому коммуникационному драйверу, что она (программа) больше не будет посылать пакеты на указанную станцию. Соответствующий драйвер освобождает виртуальный канал на уровне платы сетевого адаптера для указанного сетевого адреса.

Функцию IPXDisconnectFromTaget нельзя вызывать из программы ESR.

2.6.2. Функции AES

Если вашей программе требуется измерять временные интервалы, она может воспользоваться асинхронным планировщиком событий AES, реализованным в рамках драйвера IPX.

Для функций AES можно использовать тот же формат ECB, что и для функций IPX. Однако поля используются немного по-другому:

```
struct AES_ECB {
      void far* Link;
```

```
    void (far *ESRAddress)();
    unsigned char InUse;
    unsigned char AESWorkspace[5];
};
```

Поле AESWorkspace используется планировщиком AES. Назначение остальных полей полностью аналогично соответствующим полям обычного ECB.

IPXScheduleIPXEvent

На входе: BX = 05h.
AX = Время задержки в тиках таймера.
ES:SI = Указатель на блок ECB.

На выходе: Регистры не используются.

Функция IPXScheduleIPXEvent немедленно возвращает управление вызвавшей ее программе. После истечения временного интервала, заданного в регистре AX, поле InUse блока ECB, адрес которого задавался при вызове этой функции, сбрасывается в ноль. После этого вызывается программа ESR, если она была задана для данного ECB.

Обычно функция IPXScheduleIPXEvent используется внутри ESR, для того чтобы отложить на некоторое время обработку принятого пакета.

IPXGetIntervalMarker

На входе: BX = 08h.

На выходе: AX = Интервальный маркер.

Эта функция может использоваться для измерения временных интервалов в пределах примерно одного часа.

Возвращаемое значение - интервальный маркер - это значение, лежащее в интервале от 0000h до FFFFh и представляющее собой время в тиках таймера (следуют с интервалом примерно 1/18 секунды).

Для того, чтобы измерить время между двумя событиями, программа вызывает функцию IPXGetIntervalMarker два раза. Разность между полученными значениями является интервалом между событиями в тиках таймера.

Отметим, что вместо использования этой функции можно опрашивать значение двойного слова в области данных BIOS по адресу 0000h:046Ch. В этом слове хранится счетчик тиков таймера, значение которого обновляется каждые 55 миллисекунд.

IPXCancelEvent

На входе: BX = 06h.
ES:SI = Указатель на блок ECB.

На выходе: AL = Код завершения:
00h - функция выполнена без ошибок;
F9h - обработка ECB не может быть отменена;
FFh - указанный ECB не используется.

Функция отменяет ожидание события, связанное с указанным блоком ECB. С помощью этой функции можно отменить ожидание приема или передачи пакета, ожидание временного интервала, управляемого AES, или ожидание приема пакета SPX.

После отмены ECB поле CCode в нем устанавливается в соответствующее состояние, поле InUse устанавливается в нуль. Для отмененного ECB программа ESR не вызывается.

IPXRelinquishControl

На входе: BX = 0Ah.

На выходе: Регистры не используются.

Мы уже описывали эту функцию, предназначенную для выделения драйверу IPX процессорного времени, необходимого для его правильной работы. Приведем здесь ее еще раз, так как она по своему функциональному назначению относится к функциям асинхронного планировщика событий AES.

2.7. Определение топологии сети

Если средствами IPX или SPX необходимо передавать данные между рабочими станциями, расположенными в разных сетях, соединенных мостами, вам не обойтись без определения топологии сети. Когда программа передает данные в пределах одной сети, она должна знать адрес станции, которой будет посылаться пакет. В качестве номера сети можно указать нуль, при этом вам не надо будет даже знать номер сети, в которой расположена принимающая станция. Мы уже приводили примеры программ, передающих данные в пределах одной сети.

Другое дело, если в передачу данных вовлекается мост. Если пакет должен пройти мост, в поле ImmAddress блока ECB необходимо указать сетевой адрес моста, так как для того, чтобы попасть в другую сеть, пакет должен быть передан прежде всего в мост. В заголовке пакета при этом должен быть указан адрес принимающей станции - ее сетевой адрес, в том числе и номер сети, в которой расположена станция.

Вспомним, каким образом в приведенных ранее примерах программа-клиент и программа-сервер узнавали сетевой адрес друг друга.

Программа-клиент знала номер сокета, который используется программой-сервером. Клиент посылал пакет на этот сокет, при этом в качестве номера сети использовалось нулевое значение (пакет предназначен для передачи в пределах той сети, в которой находится передающая станция), а в качестве сетевого адреса станции - значение FFFFFFFFFFFFh (пакет предназначен для всех станций в сети). Сервер, расположенный в той же сети, что и клиент, принимал такой пакет. Анализируя поле "обратного адреса" пакета, сервер мог определить точный сетевой адрес клиента. Более того, в поле ImmAddress блока ECB, использовавшегося для приема пакета, стоял непосредственный адрес станции, от которой пришел пакет (пакет мог прийти и из другой сети через мост, в этом случае в поле ImmAddress стоял бы адрес моста).

Узнав сетевой адрес клиента, сервер отправлял ему пакет. Приняв пакет от сервера, клиент мог определить сетевой адрес сервера из заголовка пришедшего к нему пакета. Поле ImmAddress блока ECB, использовавшегося при приеме пакета от сервера, содержало непосредственный адрес станции, от которой пришел пакет.

Если сервер и клиент расположены в разных сетях, ситуация сильно усложняется. Если клиент будет посылать пакет по адресу FFFFFFFFFFFFh, указав нулевой номер сети, пакет будет принят только теми станциями, которые расположены в той же сети, что и передающая станция. Через мост такой пакет не пройдет, поэтому если программа-сервер работает на станции, которая находится в другой сети, она не получит пакет от клиента.

Для того, чтобы пакет был принят всеми станциями сети, подключенной через мост, вам необходимо послать этот пакет в мост, указав в заголовке пакета номер сети, в которую передается пакет, а также адрес станции, равный FFFFFFFFFFFFh. Для того, чтобы послать пакет в мост, в поле ImmAddress соответствующего блока ECB надо указать адрес моста.

Следовательно, для того чтобы установить связь с сервером, программа-клиент должна узнать номер сети, в которой расположен сервер, и сетевой адрес моста, через который можно послать пакет в эту сеть. К сожалению, ни одна из функций драйвера IPX или SPX не возвращает информации о конфигурации сети, поэтому ваша программа должна уметь получать такую информацию самостоятельно.

Но вы сможете выяснить конфигурацию сети, если воспользуетесь специальным *диагностическим сервисом*, реализованным в рамках драйверов протоколов IPX и SPX.

Сетевая оболочка, запущенная на рабочих станциях в сети Novell NetWare, может принимать пакеты на специальном диагностическом сокете с номером *0456h*. В ответ на принятый пакет диагностический сервис возвращает станции, пославшей такой пакет, информацию о конфигурации сетевого программного и аппаратного обеспечения станции.

Основная идея определения конфигурации сети заключается в том, что программа-клиент посылает запрос о конфигурации одновременно всем станциям данной сети на сокете 0456h, указав в качестве номера сети нуль, а в качестве адреса станции значение FFFFFFFFFFFFh. Анализируя приходящую от станций диагностическую информацию, программа-клиент может обнаружить в сети мосты и определить как номера подключенных к мостам сетей, так и сетевые адреса самих мостов.

Зная сетевой адрес мостов и номера подключенных к ним сетей, программа-клиент сможет посылать запросы для поиска программы-сервера во все подключенные к мостам сети.

Очевидно, можно посылать диагностические запросы на сокете 0456h и в другие сети с целью поиска имеющихся там мостов. Таким образом можно выяснить конфигурацию всей сети и установить связь с программой-сервером, где бы она ни находилась.

Драйверы протоколов IPX и SPX обеспечивают два вида диагностического сервиса: IPX-диагностику и SPX-диагностику. Для определения конфигурации сети нужна только IPX-диагностика. SPX-диагностика предназначена в основном для измерений производительности сети и для получения уточненной информации о составе и конфигурации программного обеспечения рабочих станций. Подробное рассмотрение SPX-диагностики выходит за рамки нашей книги.

2.7.1. Диагностический сервис IPX

Программа может посылать диагностические запросы либо конкретной станции в сети, либо всем станциям, либо всем станциям, за исключением перечисленных в списке.

Для посылки диагностического запроса программа должна подготовить IPX-пакет, состоящий из обычного заголовка размером 30 байт и блока данных, имеющего следующую структуру:

```
struct _REQ {
     unsigned char Exclusions;
     unsigned char List[80][6];
};
```

Заголовок пакета подготавливается обычным образом. В качестве номера сети можно указывать либо действительный номер сети, либо нулевое значение. В качестве сетевого адреса можно указывать либо адрес конкретной станции, либо адрес FFFFFFFFFFFFh. В поле сокета необходимо проставить значение 0456h.

В поле Exclusions блока данных необходимо проставить количество станций, от которых не требуется получать диагностику. Адреса таких станций должны быть перечислены в массиве List. Если вам надо получить диагностику от всех станций, укажите в поле Exclusions нулевое значение. В любом случае, если диагностика должна быть получена от нескольких станций, в качестве адреса в заголовке пакета необходимо указывать значение FFFFFFFFFFFFh.

Блок ECB для передачи диагностического запроса также подготавливается обычным образом. При первом диагностическом запросе в поле ImmAddress указывается значение FFFFFFFFFFFFh. В дальнейшем при определении конфигурации сети, подключенной через мост, в этом поле вы будете указывать сетевой адрес моста.

Важное замечание относительно сокета 0456h: ***вы не должны открывать или закрывать этот сокет***. Диагностический сокет уже открыт, вы должны использовать его для формирования адреса при передаче диагностического запроса. Для приема ответных пакетов конфигурации (а также для передачи запроса) вам следует динамически получить от драйвера IPX другой сокет.

После приема диагностического пакета каждая станция отвечает на него посылкой пакета конфигурации. Все эти пакеты посылаются с небольшой задержкой (примерно полсекунды), значение которой зависит от последнего байта сетевого адреса станции. Задержка используется для исключения перегрузки сети пакетами конфигурации, посылаемой одновременно многими станциями.

Послав диагностическнй пакет всем станциям, ваша программа получит несколько пакетов конфнгурации, поэтому она должна заранее (перед посылкой днагностического пакета) зарезервировать достаточное количество блоков ECB и буферов для приема пакетов конфигурации.

Принятый пакет конфигурации состоит из стандартного заголовка IPX-пакета н блока данных. Принятый блок данных состоит нз двух частей. Первая часть имеет фиксированную структуру, структура второй части зависит от конфигурацнн программного н аппаратного обеспечения станцин, от которой пришел пакет конфнгурацин.

Прнведем структуру первой части:

```
struct _RESPONSE {
     unsigned char MajorVersion;
     unsigned char MinorVersion;
     unsigned SPXDiagnosticSocket;
     unsigned char ComponentCount;
};
```

В полях **MajorVersion** и **MinorVersion** находится соответственно верхнний и нижний номер версии диагностического сервиса.

Поле **SPXDiagnosticSocket** содержит номер сокета, который должен быть нспользован для SPX-днагностики.

Самое интересное поле - **ComponentCount**. В нем находится количество компонентов программного и аппаратного обеспечення, информация о которых имеется в принятом пакете конфигурации.

Далее в принятом пакете сразу за полем ComponentCount следуют структуры, описывающие отдельные компоненты. Они могут быть двух типов - простые и расширенные. Первое поле размером в один байт имеет одинаковое значение в обоих типах структур - это идентификатор компонента. По идентификатору компонента можно однозначно судить о том, какая используется структура - простая или расширенная.

Простая структура и в самом деле несложна. Она состоит всего из одного байта идентифнкатора компонента:

```
struct _SIMPLE_COMPONENT {
     unsigned char ComponentID;
};
```

Значениями поля ComponentID для простой структуры могут быть числа 0, 1, 2, 3 илн 4:

Значение поля ComponentID	*Компонент*
0	Драйвер IPX/SPX
1	Драйвер программного обеспечения моста
2	Драйвер сетевой оболочки рабочей станцин
3	Сетевая оболочка
4	Сетевая оболочка в виде VAP-процесса

Расширенная структура сама по себе состоит из двух частей, имеющих соответственно, фиксированную и переменную структуру.

Приведем формат фиксированной части:

```
struct _EXTENDED_COMPONENT {
     unsigned char ComponentID;
     unsigned char NumberOfLocalNetworks;
};
```

Поле ComponentID может содержать значения 5, 6 или 7:

Значение поля ComponentID	*Компонент*
5	Внешний мост
6	Файл-сервер с внутренним мостом
7	Невыделенный файл-сервер

Для определения конфигурации сети важно исследовать компоненты с типом 5, 6 и 7, так как именно они имеют отношение к соединениям сетей через мосты.

Переменная часть описывает сети, подключенные к компонентам с типом 5, 6 или 7. Количество таких сетей находится в поле NumberOfLocalNetworks фиксированной части.

Для описания сетей используется массив структур (размерностью NumberOfLocalNetworks):

```
struct _NETWORK_COMPONENT {
     unsigned char NetworkType;
     unsigned char NetworkAddress[4];
     unsigned char NodeAddress[6];
};
```

Поле NetworkType описывает тип сети:

Содержимое поля NetworkType	*Тип сети*
0	Сеть, к которой подключен сетевой адаптер
1	Сеть с виртуальным сетевым адаптером (невыделенный файл-сервер)
2	Переназначенная удаленная линия (связь сетей через модемы)

Поле **NetworkAddress** содержит номер сети, к которой подключен соответствующий адаптер, а поле **NodeAddress** - сетевой адрес адаптера. Именно эти поля вам и нужны для определения номеров сетей, подключенных к мостам, и сетевых адресов самих мостов.

2.7.2. Пример программы

Приведем пример программы, которая демонстрирует способ определения конфигурации текущей сети, т. е. той сети, в которой работает данная программа. Программа создает 20 блоков ECB для приема пакетов конфигурации от

рабочих станций, ставит их в очередь на прием пакетов и посылает диагностический пакет в текущую сеть по адресу FFFFFFFFFFFFh. Затем после небольшой задержки программа анализирует блоки ECB, поставленные в очередь на прием. Если был принят пакет конфигурации, он расшифровывается и в текстовом виде выводится в стандартный выходной поток.

Приводимая ниже программа - первая программа в серии "Библиотека системного программиста", составленная на языке C++. В ней мы использовали только некоторые возможности языка C++. Для более глубокого понимания объектно-ориентированного подхода в программировании вам необходимо ознакомиться с литературой, список которой приведен в конце книги.

В функции main() создается объект класса IPX_CLIENT, который по своим функциям является клиентом. В нашем случае задача клиента - послать диагностический запрос всем станциям сети и получить от них пакет конфигурации. Можно считать, что программа диагностики, работающая на каждой станции, является сервером. Принимая запросы от клиентов на диагностическом сокете, она посылает им в ответ пакеты конфигурации.

При создании объекта класса IPX_CLIENT вызывается конструктор, выполняющий все необходимые инициализирующие действия. Конструктор инициализирует драйвер IPX, получает его точку входа и открывает динамический сокет. Соответствующий деструктор автоматически закрывает полученный сокет при завершении работы программы.

Описание класса IPX_CLIENT находится в файле ipx.hpp (см. ниже).

После того как отработает конструктор объекта IPX_CLIENT, для созданного объекта вызывается функция go(), которая и выполняет все необходимые действия. В теле функции определен массив ECB *RxECB[20] из 20 указателей на объекты класса ECB. Эти объекты используются для приема пакетов конфигурации. Кроме того, определен один объект ECB TxECB для посылки пакета с диагностическим запросом.

Программа в цикле создает 20 объектов класса ECB и с помощью функции ListenForPacket() ставит их в очередь на прием пакетов. Затем программа посылает в сеть пакет с диагностическим запросом и ждет одну секунду. За это время станции сети присылают пакеты конфигурации.

Полученные пакеты конфигурации выводятся в стандартный поток функцией PrintDiagnostics(), определенной в классе ECB. Эта функция выводит для каждой ответившей станции версию используемой диагностики, номер сокета для работы с SPX-диагностикой (не описана в нашей книге), количество программных компонентов, работающих на станции, номер сети и сетевой адрес станции (узла). Для файл-серверов и мостов дополнительно выводится количество подключенных к ним сетей. Для каждой сети выводится ее номер и сетевой адрес соответствующего адаптера.

Для упрощения программы мы ограничились диагностикой только одной сети. Кроме того, мы послали только один диагностический запрос, в котором не исключали ни одной станции. Если вам нужно определить конфигурацию всей сети, вам надо сделать следующее.

Во-первых, после приема пакетов конфигурации запишите адреса ответивших станций в список станций, исключаемых из диагностического запроса. Не забудьте проставить количество исключаемых станций. Затем выполните повторную передачу диагностического запроса. Выполняйте описанную процедуру до тех пор, пока в ответ на диагностический запрос не будет передан ни один пакет конфигурации. В этом случае адреса всех станций, имеющихся в текущей сети, будут записаны в список станций, исключаемых из диагностического запроса.

Во-вторых, выполните анализ пришедших пакетов конфигурации. Найдите пакеты, которые пришли от файл-серверов и мостов. Определите номера сетей, подключенных к ним. Затем выполните предыдущую процедуру многократной посылки диагностических пакетов в остальные сети. Для этого при передаче диагностического пакета в заголовке укажите номер сети, которую вы желаете проверить. В качестве сетевого адреса станции используйте значение FFFFFFFFFFFFh. В блоке ECB в поле непосредственного адреса укажите сетевой адрес моста, через который можно получить доступ в исследуемую сеть.

А теперь приведем текст основной программы (листинг 9):

```
// ================================================
// Листниг 9. Вызов диагностики и определение
//           конфигурации текущей сети
//
// Файл ipxdiagn.cpp
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "ipx.hpp"
// Вход в программу.
// Создаем объект - программу-клиент. Затем запускаем ее.
void main(void) {
     IPX_CLIENT NetView;
     NetView.Go();
}
// Функция определяет и распечатывает конфигурацию текущей сети.
void IPX_CLIENT::Go(void) {
// Создаем 20 ECB для приема ответов от стаиций
     ECB *RxECB[20];
// Создаем ECB для передачи диагиостического запроса.
     ECB TxECB(this->Socket, 0x456);
// Ставим заказанные ECB в очередь на прнем пакетов.
```

```
    for(int i=0; i<20; i++) {
        RxECB[i] = new ECB(this->Socket);
        RxECB[i]->ListenForPacket();
    }
// Посылаем диагностический пакет всем станциям текущей сети.
    TxECB.SendPacket();
    printf("*NetView* v1.0, (C) Фролов А.В., 1993\n"
           "Подождите немного...\n\n");
// Ждем примерно одну секунду
    sleep(1);
// Распечатываем конфигурацию сети
    printf("Конфигурация сети:\n\n");
    printf("Версия\tСокет\tКомпоненты\tСеть\t\tУзел\n");
    printf("------\t-----\t----------\t----\t\t----\n");
    for(i=0; i<20; i++) {
        RxECB[i]->PrintDiagnostics();
    }
}
```

Файл ipx.hpp содержит определения классов для приведенной выше программы (листинг 10):

```
// ================================================
// Листинг 10. Include-файл для работы с IPX
// Файл ipx.hpp
//
// (C) A. Frolov, 1993
// ================================================
#include <mem.h>
#include <dos.h>
// ----------------------
// Команды интерфейса IPX
// ----------------------
#define IPX_CMD_OPEN_SOCKET              0x00
#define IPX_CMD_CLOSE_SOCKET             0x01
#define IPX_CMD_GET_LOCAL_TARGET         0x02
#define IPX_CMD_SEND_PACKET              0x03
#define IPX_CMD_LISTEN_FOR_PACKET        0x04
#define IPX_CMD_SCHEDULE_IPX_EVENT       0x05
#define IPX_CMD_CANCEL_EVENT             0x06
#define IPX_CMD_GET_INTERVAL_MARKER      0x08
#define IPX_CMD_GET_INTERNETWORK_ADDRESS 0x09
#define IPX_CMD_RELINQUISH_CONTROL       0x0a
#define IPX_CMD_DISCONNECT_FROM_TARGET   0x0b
```

```
// ----------------------
// Коды ошибок
// ----------------------
#define NO_ERRORS           0
#define ERR_NO_IPX          1
#define ERR_NO_SPX          2
#define NO_LOGGED_ON        3
#define UNKNOWN_ERROR       0xff
// ----------------------
// Константы
// ----------------------
#define SHORT_LIVED   0
#define LONG_LIVED    0xff
#define IPX_DATA_PACKET_MAXSIZE 546
// Максимальный размер буфера данных
#define BUFFER_SIZE 512
// Внешние процедуры для инициализации и вызова драйвера IPX/SPX
extern "C" void far     ipxspx_entry(void far *ptr);
extern "C" int          ipx_init(void);
extern unsigned         IntSwap(unsigned i);
void IPXRelinquishControl(void);
// Структура для вызова драйвера IPX/SPX
struct IPXSPX_REGS {
          unsigned int    ax;
          unsigned int    bx;
          unsigned int    cx;
          unsigned int    dx;
          unsigned int    si;
          unsigned int    di;
          unsigned int    es;
};
// Класс динамических сокетов
class DYNAMIX_SOCKET {
public:
     unsigned errno;
     unsigned Socket;
     struct IPXSPX_REGS iregs;
// Конструктор динамического сокета.
// Открывает сокет и запоминает его номер.
     DYNAMIX_SOCKET() {
          iregs.bx = IPX_CMD_OPEN_SOCKET;
          iregs.dx = 0;
          iregs.ax = 0;
          ipxspx_entry( (void far *)&iregs );
```

```
            Socket = iregs.dx;
            errno = iregs.ax;
      };
// Деструктор. Закрывает ранее открытый сокет.
      ~DYNAMIX_SOCKET() {
            iregs.bx = IPX_CMD_CLOSE_SOCKET;
            iregs.dx = Socket;
            ipxspx_entry( (void far *)&iregs );
      };
};
// Класс программ-клиентов IPX
class IPX_CLIENT {
public:
      unsigned errno;
// Сокет, с которым работает программа-клиент
      DYNAMIX_SOCKET *Socket;
// Конструктор. Выполняет инициализацию клиента:
// инициализирует драйвер IPX и открывает динамический сокет.
      IPX_CLIENT() {
            if(ipx_init() != 0xff) {
                  errno = 0xff; return;
            }
            Socket = new DYNAMIX_SOCKET;
      }
// Деструктор. Автоматически закрывает
// сокет при завершении работы программы.
      ~IPX_CLIENT() {
            delete Socket;
      }
// Фуикция, определяющая конфигурацию сети
      void Go(void);
};
// Класс заголовков IPX-пакетов.
struct IPX_HEADER {
// Структура, описывающая заголовок
      struct _IPX_HEADER {
            unsigned int    Checksum;
            unsigned int    Length;
            unsigned char   TransportControl;
            unsigned char   PacketType;
            unsigned char   DestNetwork[4];
            unsigned char   DestNode[6];
            unsigned int    DestSocket;
```

```
            unsigned char   SourceNetwork[4];
            unsigned char   SourceNode[6];
            unsigned int    SourceSocket;
      } _ipx_header;
// Конструктор. Записывает в заголовок тип пакета,
// нулевой номер сети, в которую будет отправлен пакет,
// адрес 0xFFFFFFFFFFFF в качестве адреса назначения,
// номера сокетов адресата и отправителя пакета.
      IPX_HEADER(unsigned Socket, unsigned SrcSocket) {
            _ipx_header.PacketType = 4;
            memset(_ipx_header.DestNetwork, 0, 4);
            memset(_ipx_header.DestNode, 0xff, 6);
            _ipx_header.DestSocket = Socket;
            _ipx_header.SourceSocket = SrcSocket;
      }
// Конструктор. Записывает в заголовок тип пакета,
// нулевой номер сети, в которую будет отправлен пакет,
// адрес 0xFFFFFFFFFFFF в качестве адреса назначения.
      IPX_HEADER() {
            _ipx_header.PacketType = 4;
            memset(_ipx_header.DestNetwork, 0, 4);
            memset(_ipx_header.DestNode, 0xff, 6);
      }
};
// Класс блоков ECB.
struct ECB {
// Сам блок ECB в стандарте IPX/SPX.
            struct _ECB {
                  void far   *Link;
                  void far   (*ESRAddress)(void);
                  unsigned char   InUse;
                  unsigned char   CCode;
                  unsigned int    Socket;
                  unsigned int    ConnectionId;
                  unsigned int    RrestOfWorkspace;
                  unsigned char   DriverWorkspace[12];
                  unsigned char   ImmAddress[6];
                  unsigned int    FragmentCnt;
                  struct {
                        void far   *Address;
                        unsigned int Size;
                  } Packet[2];
            } _ecb;
// Указатель на заголовок пакета, связанного с данным ECB.
            struct IPX_HEADER *IPXHeader;
```

```
// Структура для приема ответа от стаиции
// после посылки диагностического пакета.
          struct Reply {
               unsigned char MajVer;
               unsigned char MinVer;
               unsigned Socket;
               unsigned char NumberOfComponents;
               unsigned char Buffer[512];
          } Rep;
// Структура для хранения диагиостического пакета.
          struct DiagnRequest {
               unsigned char Exclusions;
               unsigned char List[80][6];
          } DReq;
          struct IPXSPX_REGS iregs;
// Конструктор. Создается заголовок пакета,
// в блок ECB записывается номер сокета, используемого клиентом,
// инициализируются счетчик фрагментов и дескрипторы фрагментов.
// В качестве непосредственного адреса указывается
// адрес 0xFFFFFFFFFFFF.
          ECB(DYNAMIX_SOCKET *Socket) {
               IPXHeader = new IPX_HEADER;
               memset(&_ecb, 0, sizeof(_ecb));
               _ecb.Socket = Socket->Socket;
               _ecb.FragmentCnt       = 2;
               _ecb.Packet[0].Address = &(IPXHeader->_ipx_header);
               _ecb.Packet[0].Size    = 30;
               _ecb.Packet[1].Address = &Rep;
               _ecb.Packet[1].Size    = sizeof(Rep);
               memset(_ecb.ImmAddress, 0xff, 6);
          }
// Конструктор. Создается заголовок пакета, в блок ECB записывается
// номер сокета, используемого клиентом, а также номер сокета
// адресата, ннициалнзируются счетчик фрагментов и дескрипторы
// фрагментов. В качестве непосредственного адреса указывается
// адрес 0xFFFFFFFFFFFF.
          ECB(DYNAMIX_SOCKET *Socket, unsigned DstSocket) {
               IPXHeader = new IPX_HEADER(IntSwap(DstSocket),
                    Socket->Socket);
// Запрос адресуется всем станциям без исключения.
               DReq.Exclusions = 0;
               memset(&_ecb, 0, sizeof(_ecb));
               _ecb.Socket = Socket->Socket;
               _ecb.FragmentCnt       = 2;
               _ecb.Packet[0].Address = &(IPXHeader->_ipx_header);
```

```
            _ecb.Packet[0].Size    = 30;
            _ecb.Packet[1].Address = &DReq;
            _ecb.Packet[1].Size    = sizeof(DReq);
            memset(_ecb.ImmAddress, 0xff, 6);
        }
// Прием IPX-пакета.
        void ListenForPacket(void) {
            iregs.es = FP_SEG((void far*)&_ecb);
            iregs.si = FP_OFF((void far*)&_ecb);
            iregs.bx = IPX_CMD_LISTEN_FOR_PACKET;
            ipxspx_entry( (void far *)&iregs );
        }
// Передача IPX-пакета.
        void SendPacket(void) {
            iregs.es = FP_SEG((void far*)&_ecb);
            iregs.si = FP_OFF((void far*)&_ecb);
            iregs.bx = IPX_CMD_SEND_PACKET;
            ipxspx_entry( (void far *)&iregs );
        }
// Распечатать принятый пакет конфигурации
        void PrintDiagnostics(void);
};
```

В файл ipx.cpp (листинг 11) мы вынесли остальные используемые программой функции, в частности функцию PrintDiagnostics(). Кроме того, программа вызывает функции, определенные в файле ipxdrv.asm, содержимое которого уже было приведено нами раньше.

```
// ================================================
// Листинг 11. Функции IPX.
//
// Файл ipx.cpp
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include "ipx.hpp"
/**
* .Name    IntSwap
*
* .Title   Обмен байтов в слове
*
* .Descr   Функция меняет местами байты в слове,
```

```
*          которое передается ей в качестве параметра
*
* .Params  unsigned i - преобразуемое слово
*
* .Return  Преобразованное слово
**/
unsigned IntSwap(unsigned i) {
     return((i>>8) | (i & 0xff)<<8);
}
/**
* .Name    IPXRelinquishControl
*
* .Title   Передать управление IPX при ожидании
*
* .Descr   Фуикция используется при ожидаии
*          завершеиия приема через опрос поля InUse
*          блока ECB.
*
* .Params  Не используются
*
* .Return  Ничего
**/
void IPXRelinquishControl(void) {
     struct IPXSPX_REGS iregs;
     iregs.bx = IPX_CMD_RELINQUISH_CONTROL;
     ipxspx_entry( (void far *)&iregs );
}
// Функция для печати содержимого прииятого пакета конфигурации.
void ECB::PrintDiagnostics(void) {
     int i, j, k, networks, component;
// Печатаем конфигурацию только для тех ECB,в поле InUse которых
// стонт нулевое значение, т.е. если был принят пакет.
          if(!_ecb.InUse) {
// Распечатываем версию диагиостической поддержки, номер сокета для
// SPX-диагностики и количество компоиентов программного
//обеспечения, работающего иа стаиции.
               printf("\n%d.%d\t%d\t%d\t\t",
                    Rep.MajVer, Rep.MinVer,
                    Rep.Socket, Rep.NumberOfComponents);
// Распечатываем номер сети, из которой пришел пакет конфигурации.
               for(i=0;i<4;i++) {
                 printf("%02.2X",(unsigned char)
                    IPXHeader->_ipx_header.SourceNetwork[i]);
               }
               printf("\t");
```

```
// Распечатываем сетевой адрес станции, из
// которой пришел пакет конфигурации.
                for(i=0;i<6;i++) {
                  printf("%02.2X",(unsigned char)
                     IPXHeader->_ipx_header.SourceNode[i]);
                }
                printf("\n\n");
// Для каждого программного компонента распечатываем его название.
                for(i=0;i<Rep.NumberOfComponents;) {
                     switch(component=Rep.Buffer[i]) {
                     case 0:
                          printf("\tДрайвер IPX/SPX\n");
                          i++;
                          break;
                     case 1:
                          printf("\tДрайвер моста\n");
                          i++;
                          break;
                     case 2:
                          printf("\tДрайвер сетевой оболочкн\n");
                          i++;
                          break;
                     case 3:
                          printf("\tСетевая оболочка\n");
                          i++;
                          break;
                     case 4:
                          printf("\tОболочка VAP\n");
                          i++;
                          break;
// Для мостов н серверов дополнительно выводим количество
// подключенных к ним сетей,тип каждой сетн, номера подключеиных
// сетей и сетевые адреса адаптеров.
                     case 5: case 6: case 7:
                          switch(component) {
                          case 5:
                               printf("\tВыделенный мост\n");
                               break;
                          case 6:
                               printf("\tФайл-сервер/внутренний
                                         мост\n");
                               break;
                          case 7:
                               printf("\tНевыделенный сервер\n");
                               break;
                          }
                          i++;
```

```
// Количество подключенных сетей
                        printf("\t\tПодключено сетей: %d",
                             (unsigned char)Rep.Buffer[i]);
                        networks = Rep.Buffer[i];
                        i++;
// Для каждой сети печатаем ее тип,
// номер сети и сетевой адрес адаптера.
                        for(j=0;j<networks;j++) {
// Тип сети
                             printf("\n\t\t\tТип сети: %d\t",
                                  (unsigned char)
                                       Rep.Buffer[i++]);
// Номер сети
                             for(k=0;k<4;k++,i++) {
                               printf("%02.2X",(unsigned char)
                                  Rep.Buffer[i]);
                             }
                             printf("\t");
// Сетевой адрес адаптера
                             for(k=0;k<6;k++,i++) {
                               printf("%02.2X",(unsigned char)
                                  Rep.Buffer[i]);
                             }
                        }
                        printf("\n");
                        break;
                   }
              }
         }
}
```

Приведем образец листинга, выдаваемого программой в стандартный поток вывода:

```
*NetView* v1.0, (C) Фролов А.В., 1993
Подождите немного...
Конфигурация сети:
Версия  Сокет    Компоненты              Сеть         Узел
------  -----    ----------              ----         ----
1.0     576      3                       00000010     000000000001
                 Драйвер IPX/SPX
                 Драйвер моста
                 Файл-сервер/внутренний мост
                 Подключено сетей: 3
                    Тип сети: 1          00000010     000000000001
                    Тип сети: 0          00000013     48450000456C
```

```
                  Тип сети: 0          00000012     4845000047C7
1.0     576       3                    0000000E     000000000001
                  Драйвер IPX/SPX
                  Драйвер моста
                  Файл-сервер/внутренний мост
                  Подключено сетей: 2
                     Тип сети: 1       0000000E     000000000001
                     Тип сети: 0       00000012     008428801E9D
1.1     320       3                    00000012     008058801C82
                  Драйвер IPX/SPX
                  Драйвер сетевой оболочки
                  Сетевая оболочка
1.1     320       3                    00000012     484500004666
                  Драйвер IPX/SPX
                  Драйвер сетевой оболочки
                  Сетевая оболочка
1.0     320       3                    00000012     484500004889
                  Драйвер IPX/SPX
                  Драйвер сетевой оболочки
                  Сетевая оболочка
1.1     320       3                    00000012     008058801F0F
                  Драйвер IPX/SPX
                  Драйвер сетевой оболочки
                  Сетевая оболочка
1.1     320       3                    00000012     000561E5D284
                  Драйвер IPX/SPX
                  Драйвер сетевой оболочки
                  Сетевая оболочка
1.1     320       3                    00000012     008058801E1D
                  Драйвер IPX/SPX
                  Драйвер сетевой оболочки
                  Сетевая оболочка
1.1     320       3                    00000012     484506004726
                  Драйвер IPX/SPX
                  Драйвер сетевой оболочки.
                  Сетевая оболочка
1.1     320       3                    00000012     008058801EB5
                  Драйвер IPX/SPX
                  Драйвер сетевой оболочки
                  Сетевая оболочка
1.0     320       3                    00000012     484556004705
                  Драйвер IPX/SPX
```

```
Драйвер сетевой оболочки
Сетевая оболочка
```

Из приведенного листинга видно, что в сети имеются два файл-сервера и 9 рабочих станций. Все рабочие станции находятся в сети 00000012. К этой же сети подключены оба файл-сервера. Первый в списке файл-сервер подключен к сетям 00000012 и 00000013, следовательно, этот файл-сервер является внутренним мостом между сетью 00000012 и 00000013.

Обратите внимание, что для двух файл-серверов, имеющихся в сети, указан тип сети 1 и номера сетей 10h и 0Eh. Это так называемые внутренние номера сетей, которые задавались при генерации Novell NetWare версии 3.11. Физические сети имеют в нашем случае номера 12 и 13.

2.8. Настройка параметров IPX

Мы уже говорили, что драйвер протоколов IPX/SPX для MS-DOS реализован в виде резидентной программы. В версии 3.11 операционной системы Novell NetWare на рабочих станциях MS-DOS используется программа ipxodi.com.

При запуске этой программы вы можете указать параметры "d" и "a". Если указывается параметр "d", на рабочей станции не загружается диагностический сервис, что экономит примерно 4 Кбайт памяти. Если же указывается параметр "a", в память загружается только драйвер протокола IPX, а драйвер протокола SPX и диагностический сервис не загружаются. При этом освобождается 8 Кбайт основной памяти.

Однако учтите, что такие сетевые утилиты, как RCONSOLE и NVER, требуют присутствия драйвера протокола SPX и диагностического сервиса.

Заметим также, что вы можете менять некоторые параметры драйверов IPX и SPX. Для этого в первых строках файла net.cfg, расположенного в каталоге C:\NET (см. предыдущий том "Библиотеки системного программиста") можно указывать параметры:

IPX RETRY COUNT — Параметр определяет, сколько раз будет выполнена повторная передача пакета, прежде чем будет сделан вывод о невозможности его передачи. Сам протокол IPX не выполняет повторные передачи (так как этот протокол не гарантирует доставку передаваемых пакетов), но это значение используется протоколами более высокого уровня, реализованными на базе IPX, в частности протоколом SPX.
По умолчанию пакет передается 20 раз.

IPX SOCKETS — Параметр определяет максимальное количество сокетов, которые программа может открыть на рабочей станции.
По умолчанию можно открыть 20 сокетов.

Например, для увеличения числа доступных сокетов до 50 добавьте в начало файла net.cfg строку `IPX SOCKETS=50`.

Составляя документацию по установке разрабатываемого вами сетевого программного обеспечения, не забудьте упомянуть о том, как надо настраивать параметры протокола IPX (если такая настройка необходима для нормальной работы вашей программы).

Глава 3

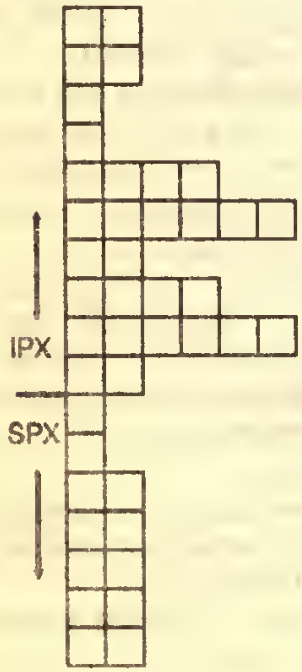
ПРОТОКОЛ SPX

Для некоторых приложений (например, для программ, передающих файлы между рабочими станциями) удобнее использовать сетевой протокол более высокого уровня, обеспечивающий гарантированную доставку пакетов в правильной последовательности. Разумеется, ваша программа может сама следить за тем, чтобы все переданные пакеты были приняты. Однако в этом случае вам придется делать собственную надстройку над протоколом IPX - собственный протокол передачи данных.

Прежде чем принять решение о создании собственного протокола, изучите протокол SPX - протокол последовательного обмена пакетами (Sequenced Packet Exchange Protocol), разработанный Novell. Возможно, что протокол SPX удовлетворит потребности вашей программы в гарантированной передаче данных по сети.

3.1. Формат пакета SPX

Пакет, передаваемый при помощи протокола SPX, имеет более длинный заголовок. Дополнительно к 30 байтам стандартного заголовка пакета IPX добавляется еще 12 байт (рис. 4).



Checksum - контрольная сумма
Length - общая длина пакета
TransportControl - счетчик пройденных мостов
PacketType - тип пакета
DestNetwork - номер сети получателя пакета
DestNode - адрес станции-получателя
DestSocket - сокет программы-получателя
SourceNetwork - номер сети отправителя пакета
SourceNode - адрес станции-отправителя
SourceSocket - сокет программы-отправителя
ConnControl - управление потоком данных
DataStreamType - тип данных в пакете
SourceConnID - идентификатор канала отправителя
DestConnID - идентификатор канала получателя
SeqNumber - счетчик переданных пакетов
AckNumber - номер следующего пакета
AllocNumber - количество буферов для приема

Рис. 4. Формат заголовка пакета SPX

Приведем структуру заголовка пакета SPX для использования в программах, составленных на языке Си:

```
struct SPX_HEADER {
          unsigned int    Checksum;
          unsigned int    Length;
          unsigned char   TransportControl;
          unsigned char   PacketType;
          unsigned char   DestNetwork[4];
          unsigned char   DestNode[6];
          unsigned int    DestSocket;
          unsigned char   SourceNetwork[4];
          unsigned char   SourceNode[6];
          unsigned int    SourceSocket:
// ------------Специфическая для SPX часть ---------
          unsigned char  ConnControl;
          unsigned char  DataStreamType;
          unsigned char  SourceConnID[2];
          unsigned char  DestConnID[2];
          unsigned char  SequenceNumber[2];
          unsigned char  AckNumber[2];
          unsigned char  AllocationNumber[2];
};
```

Поле **ConnControl** можно рассматривать как набор битовых флагов, управляющих передачей данных по каналу SPX:

Биты	*Назначение*
01h-08h	Зарезервировано
10h	End-of-Message. Этот бит может использоваться программой для сигнализации окончания передачи данных. Драйвер SPX передает этот бит программе в неизменном виде, причем сам драйвер протокола SPX этот бит игнорирует
20h	Attention. Этот бит игнорируется драйвером SPX и передается в неизменном виде программе
40h	Acknowledgement Required. Бит используется драйвером SPX. Вам не следует модифицировать его значение
80h	System Packet. Этот бит устанавливается драйвером SPX при передаче системных пакетов, которые используются самим драйвером и не передаются в программу пользователя

Поле **DataStreamType** также состоит из однобитовых флагов, которые используются для классификации данных, передаваемых или принимаемых при помощи протокола SPX. Приведем возможные значения поля DataStreamType:

Биты	*Назначение*
00h-FDh	Эти значения игнорируются драйвером SPX и могут быть использованы программой произвольным образом
FEh	End-of-Connection. Когда программа вызывает функцию, закрывающую SPX-канал, драйвер SPX посылает партнеру по связи последний пакет, в поле DataStreamType которого записано значение FEh. Это служит требованием завершить связь и закрыть канал
FFh	End-of-Connection-Acknowledgement. Это значение отмечает пакет, подтверждающий завершение связи. Такой пакет является системным и не передается в программу пользователя

Поле **SourceConnID** содержит номер канала связи передающей программы, присвоенный драйвером SPX при создании канала связи. Этот номер должен указываться функции передачи пакета средствами SPX.

Поле **DestConnID** содержит номер канала связи принимающей стороны. Так как все пакеты приходят на один номер сокета и могут принадлежать разным каналам связи (на одном сокете можно открыть несколько каналов связи), вам необходимо классифицировать приходящие пакеты по номеру канала связи.

Поле **SeqNumber** содержит счетчик пакетов, переданных по каналу в одном направлении. На каждой стороне канала используется свой счетчик. После достижения значения FFFFh счетчик сбрасывается в нуль, после чего процесс счета продолжается.

Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.

Поле **AckNumber** содержит номер следующего пакета, который должен быть принят драйвером SPX.

Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.

Поле **AllocNumber** содержит количество буферов, распределенных программой для приема пакетов.

Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.

3.2. Блок ECB

Для протокола SPX используется точно такой же блок ECB, что и для протокола IPX.

3.3. Функции SPX

3.3.1. Инициализация SPX

SPXCheckInstallation

На входе: BX = 10h.
AL = 00h.

На выходе: AL = Код завершения:
00h - SPX не установлен;
FFh - SPX установлен.
BH = Верхний (major) номер версии SPX.
BL = Нижний (minor) номер версии SPX.
CX = Максимальное количество каналов SPX, поддерживаемых драйвером SPX.
DX = Количество доступных каналов SPX.

Прежде чем использовать функции SPX, программа должна вызвать функцию SPXCheckInstallation для того, чтобы убедиться в наличии драйвера SPX.

3.3.2. Образование канала связи

SPXListenForConnection

На входе: BX = 12h.
AL = Счетчик повторов попыток создать канал связи.
AH = Флаг включения системы периодической проверки связи (Watchdog Supervision Required Flag).
ES:SI = Указатель на блок ECB.

На выходе: Регистры не используются.

Эта функция используется в паре с функцией SPXEstablishConnection для образования канала связи.

Программа-сервер вызывает SPXListenForConnection, передавая ей адрес блока ECB. Этот блок будет использован для образования канала связи. Когда программа-клиент вызовет функцию SPXEstablishConnection, произойдет образование канала связи и в поле InUse блока ECB будет записано нулевое значение. Будет также вызвана соответствующая программа ESR, если задан ее адрес.

В блоке ECB необходимо определить значение поля ESRAddress и указать номер сокета. Для каждого сокета вы можете образовать несколько каналов.

Блок ECB, адрес которого задан в регистрах ES:SI, ставится драйвером SPX во внутреннюю очередь блоков ECB, ожидающих прихода пакетов от функций SPXEstablishConnection, выдаваемых другими станциями, желающими образовать канал связи.

После образования канала связи, когда в поле InUse будет записано нулевое значение, поле CCode блока ECB будет содержать код завершения:

00 канал связи создан, ошибок нет;
FFh указанный в ECB сокет не был открыт;
FCh запрос SPXListenForConnection был отменен функциями IPXCancelEvent или IPXCloseSocket (ESR не вызывается);
EFh переполнилась внутренняя таблица номеров каналов связи; до тех пор, пока какой-нибудь канал не будет закрыт, вы не сможете образовать новые каналы.

Перед вызовом функции SPXListenForConnection программа должна выделить хотя бы один ECB для приема SPX-пакета. Это нужно сделать при помощи функции SPXListenForSequencedPacket (см. ниже описание функции).

Вам также надо задать в регистре AL счетчик повторов попыток создания канала связи и в регистре AH - флаг включения системы периодической проверки связи. Вы можете задать от 1 до 255 попыток или использовать значение по умолчанию, если запишете в регистр AL нулевое значение.

Если в регистр AH будет записано ненулевое значение, драйвер SPX будет периодически проверять работоспособность канала связи, передавая специальные тестовые пакеты. Если канал вдруг перестает работать, он разрывается и в ECB, подготовленный для приема пакетов функцией SPXListenForSequencedPacket, в поле InUse проставляется нулевое значение. Поле CCode при этом будет содержать код ошибки EDh, а номер "испорченного" канала будет записан в первых двух байтах поля IPXWorkspace блока ECB.

SPXEstablishConnection

На входе: BX = 11h.
AL = Счетчик повторов попыток создать канал связи.
AH = Флаг включения системы периодической проверки связи (Watchdog Supervision Required Flag).
ES:SI = Указатель на блок ECB.

На выходе: AL = Промежуточный код завершения:
00h - выполняется попытка создать канал;
FFh - указанный в блоке ECB сокет закрыт;
FDh - сбойный пакет: либо счетчик фрагментов не равен 1, либо размер фрагмента не равен 42;
EFh - переполнение локальной таблицы номеров каналов связи.
DX = Присвоенный номер канала.

Функция устанавливает канал связи с программой, предварительно вызвавшей функцию SPXListenForConnection.

Для функции необходимо подготовить блок ECB и пакет в формате SPX, состоящий из одного заголовка. В блоке ECB необходимо заполнить поля ESRAddress, Socket, счетчик количества фрагментов (нужен один фрагмент) и

указатель на фрагмент размером 42 байта. В заголовке SPX-пакета необходимо заполнить поля DestNetwork, DestNode, DestSocket.

Кроме того, перед вызовом функции SPXListenForConnection программа должна выделить хотя бы один ECB для приема SPX-пакета. Это нужно сделать при помощи функции SPXListenForSequencedPacket (см. ниже описание функции).

Канал создается в два приема.

На первом этапе проверяется возможность образования канала - проверяется наличие свободного места в таблице номеров каналов, проверяется таблица сокетов, размер пакета. Если все хорошо, с целью попытки создать канал удаленному партнеру посылается пакет, после чего функция возвращает управление вызвавшей ее программе. Регистр AL при этом содержит промежуточный код завершения. Если этот код равен нулю, можно переходить к ожиданию приема ответного пакета от партнера по созданию канала. Регистр DX при этом содержит номер присвоенного канала.

Если партнер отвечает соответствующим образом, в поле InUse блока ECB устанавливается нулевое значение. Если при этом в поле CCode также находится нулевое значение, канал считается созданным.

Номер канала удаленного партнера, который вы будете использовать для передачи ему пакетов функцией SPXSendSequencedPacket, находится в поле SourceConnID блока ECB. Сохраните его для дальнейшего использования.

Если по каким-либо причинам канал создать не удалось, в поле CCode будет записан код ошибки:

00h канал связи создан, ошибок нет;

FCh запрос SPXListenForConnection был отменен функциями IPXCancelEvent или IPXCloseSocket (ESR не вызывается);

FDh сбойный пакет: либо счетчик фрагментов не равен единице, либо размер фрагмента не равен 42;

FFh указанный в ECB сокет не был открыт;

EFh переполнилась внутренняя таблица номеров каналов связи; до тех пор, пока какой-нибудь канал не будет закрыт, вы не сможете образовать новые каналы;

EDh адресат не отвечает или сообщает, что он не может создать канал; этот код может возникнуть либо как результат неисправности сетевого аппаратного обеспечения, либо если функция SPXEstablishConnection была отменена при помощи функции SPXAbortConnection.

Обратим ваше внимание на то, что для отмены создания канала необходимо пользоваться специально предназначенной для этого функцией SPXAbortConnection, а не функцией IPXCancelEvent.

3.3.3. Прием и передача пакетов

SPXListenForSequencedPacket

На входе: BX = 17h.
ES:SI = Указатель на блок ECB.

На выходе: Регистры не используются.

Функция обеспечивает прием пакетов средствами протокола SPX. При этом она ставит блок ECB, адрес которого передается через регистры ES:SI, в очередь на прием, после чего немедленно возвращает управление вызвавшей программе.

После того как пакет будет принят, в поле InUse блока ECB устанавливается нулевое значение, а в поле CCode - код завершения:

00h пакет принят без ошибок;
FCh запрос SPXListenForSequencedPacket был отменен функциями IPXCancelEvent или IPXCloseSocket (ESR не вызывается);
FDh переполнение пакета - принятый пакет имеет длину, которая превосходит размер буферов, указанных в дескрипторах фрагментов;
EDh система периодической проверки связи обнаружила разрыв канала, номер разрушенного канала записан в первых двух байтах поля IPXWorkspace блока ECB;
FFh указанный в ECB сокет не был открыт.

Перед вызовом функции необходимо заполнить в ECB поля ESRAddress, Socket, счетчик фрагментов и дескрипторы фрагментов. При этом первый фрагмент передаваемого пакета должен иметь длину не менее 42 байт - это буфер для приема стандартного заголовка SPX-пакета. Необходимо также открыть используемый сокет при помощи функции IPXOpenSocket.

Обычно для приема пакетов используется несколько блоков ECB. Все они последовательно ставятся в очередь функцией SPXListenForSequencedPacket. Для приема пакета драйвером SPX может быть использован любой свободный блок ECB. Не гарантируется, что блоки ECB будут использованы именно в том порядке, в котором они ставились в очередь функцией SPXListenForSequencedPacket.

Если принимается системный пакет, использованный блок ECB автоматически возвращается в очередь для приема пакетов.

Так как для обработки системных пакетов протокол гарантированной доставки SPX использует те же блоки ECB, что и для приема прикладных пакетов, ваша программа должна обеспечить достаточное количество блоков ECB в очереди на прием пакетов.

Если принимается пакет, у которого в заголовке в поле DataStreamType находится значение FEh, это означает, что передающая программа собирается завершить передачу и закрыть канал. При этом все блоки ECB, стоящие в очередь на передачу пакетов (в которую они ставятся функцией SPXSendSequencedPacket, описанной ниже), отмечаются нулевым значением в поле InUse и соответствующим кодом завершения в поле CCode.

Программа может отменить ожидание завершения приема пакета для блока ECB при помощи функции IPXCancelEvent, при этом она должна *заново проинициализировать* поле ESRAddress перед повторным использованием этого блока ECB.

SPXSendSequencedPacket

На входе: BX = 16h.
ES:SI = Указатель на блок ECB
DX = Номер канала связи.

На выходе: --- Регистры не используются.

Функция ставит блок ECB, адрес которого указан в регистрах ES:SI, в очередь на передачу, после чего немедленно возвращает управление вызвавшей программе.

Перед вызовом функции программа должна заполнить поле ESRAddress, счетчик фрагментов и дескрипторы фрагментов блока ECB, а также бит End-Of-Message в поле ConnControl и поле DataStreamType в заголовке передаваемого пакета. Разумеется, заголовок должен иметь длину 42 байта.

В регистр DX необходимо загрузить номер канала, используемый партнером.

В отличие от средств передачи пакета протокола IPX успешное завершение передачи пакета, инициированной функцией SPXSendSequencedPacket, гарантирует доставку пакета партнеру. Если партнер не успевает принимать передаваемые пакеты, они ставятся в очередь на передачу, чем обеспечивается правильная последовательность доставки пакетов.

После завершения передачи пакета поле InUse блока ECB имеет нулевое значение. Если определена программа ESR, она вызывается. В поле CCode находится код завершения:

00h пакет был передан *и успешно принят* партнером;
FCh указанный в ECB сокет был закрыт, программа ESR не вызывается;
FDh сбойный пакет: либо счетчик фрагментов равен нулю, либо размер первого фрагмента меньше 42 байт, либо размер всего пакета больше 576 байт;
EEh неправильное значение в регистре DX;
EDh либо система периодической проверки связи обнаружила разрыв канала, либо канал был уничтожен функцией SPXAbortConnection (номер разрушенного канала записан в первых двух байтах поля IPXWorkspace блока ECB);
ECh удаленный партнер закрыл канал без подтверждения приема этого пакета, при этом SPX не может гарантировать, что переданный пакет был успешно принят партнером перед тем, как канал был закрыт.

Для отмены передачи пакета нельзя использовать функцию IPXCancelEvent. Вместо нее необходимо использовать функцию SPXAbortConnection.

3.3.4. Разрыв канала связи

SPXTerminateConnection

На входе: BX = 13h.
ES:SI = Указатель на блок ECB.
DX = Номер канала связи.

На выходе: Регистры не используются.

Функция посылает удаленному партнеру пакет, который состоит из одного заголовка. В поле DataStreamType этого заголовка находится значение FEh, которое говорит партнеру о том, что необходимо закрыть канал. Сразу после вызова функция возвращает управление вызывавшей ее программе.

Перед вызовом функции программа должна заполнить поле ESRAddress, счетчик фрагментов (в пакете должен быть один фрагмент размером 42 байта) и дескриптор фрагмента блока ECB.

В регистр DX необходимо загрузить номер канала, используемый партнером.

После завершения процесса закрытия канала в поле InUse блока ECB проставляется нулевое значение и вызывается программа ESR (если она была задана). В поле CCode проставляется код завершения:

00h канал был успешно закрыт;
FDh сбойный пакет: либо счетчик фрагментов не равен единице, либо размер фрагмента меньше 42 байт;
EEh неправильное значение в регистре DX;
EDh канал закрылся с ошибкой, при этом удаленный партнер не прислал пакет, подтверждающий закрытие канала. При этом SPX не гарантирует, что партнер успешно закрыл канал со своей стороны;
ECh удаленный партнер закрыл канал без подтверждения команды закрытия канала, при этом SPX не может гарантировать, что партнер вызвал функцию, закрывающую канал.

После закрытия канала освобождается место в таблице номеров каналов. Программа может открывать новые каналы.

Заметим, что для отмены ожидания завершения процесса закрытия канала необходимо использовать функцию SPXAbortConnection, а не IPXCancelEvent.

SPXAbortConnection

На входе: BX 14h.
DX Номер канала связи.

На выходе: Регистры не используются.

Функция разрывает канал связи без "согласования" с партнером. Данная функция должна использоваться только в катастрофических случаях, когда невозможно выполнить нормальную процедуру закрытия канала.

После вызова этой функции во всех ECB, относящихся к данному каналу в поле CCode проставляется значение EDh.

3.3.5. Проверка состояния канала

SPXGetConnectionStatus

На входе: BX = 15h.
DX = Номер канала связи.
ES:SI = Указатель на буфер размером 44 байта.

На выходе: AL = Код завершения:
00h - канал активен;
EEh - указанный канал не существует.

С помощью функции SPXGetConnectionStatus программа может проверить состояние канала. Если канал существует, в буфер, адрес которого задан в регистрах ES:SI, записывается информация о состоянии канала.

Приведем формат буфера в виде структуры:

```
struct CSB {
     unsigned char ConnectionState;
     unsigned char ConnectionFlags;
     unsigned char SrcConnectionID[2];
     unsigned char DestConnectionID[2];
     unsigned char SeqNumber[2];
     unsigned char AckNumber[2];
     unsigned char AllocNumber[2];
     unsigned char RemoteAckNumber[2];
     unsigned char RemoteAllocNumber[2];
     unsigned char ConnectionSocket[2];
     unsigned char ImmAddress[6];
     unsigned char DestNetwork[4];
     unsigned char DestNode[6];
     unsigned char DestSocket[2];
     unsigned char RetransmissionCount[2];
     unsigned char EstimatedRoundtripDelay[2];
     unsigned char RetransmittedPackets[2];
     unsigned char SuppressedPackets[2];
};
```

Все поля в этой структуре имеют "перевернутый" формат, в котором младшие байты записаны по старшему адресу.

Поле **ConnectionState** отображает текущее состояние канала:

01h драйвер SPX находится в состоянии ожидания приема пакета, посылаемого функцией SPXEstablishConnection;

02h драйвер SPX пытается создать канал с удаленной рабочей станцией после вызова функции SPXEstablishConnection;

03h канал создан;

04h канал закрыт.

Поле **ConnectionFlags** содержит флаги, которые используются драйвером SPX для управления каналом. Бит 02h, в частности, управляет использованием системы периодической проверки связи. Если этот бит установлен в единицу, для данного канала выполняется периодическая проверка связи.

Поле **SrcConnectionID** содержит номер канала, присвоенный локальной станции. Это тот самый номер канала, который надо загружать в регистр DX перед использованием функций SPX.

Поле **DestConnectionID** содержит номер канала, присвоенный программе, работающей на удаленной станции.

Поле **SeqNumber** содержит последовательный номер, который SPX будет использовать для пересылки следующего пакета по каналу.

Поле **AckNumber** содержит последовательный номер следующего пакета, который должен быть принят по каналу от удаленной станции.

Поле **AllocNumber** используется драйвером SPX для контроля за пакетами, которые были переданы, но для которых еще не пришло подтверждение о приеме. В нем содержится количество свободных буферов, распределенных для приема пакетов.

Поле **RemoteAckNumber** содержит номер следующего пакета, который должен быть принят на удаленной станции от локальной станции.

Поле **RemoteAllocNumber** имеет назначение, аналогичное назначению поля AllocNumber, но относится к удаленной станции.

Поле **ConnectionSocket** содержит номер сокета, который используется драйвером SPX для приема и передачи пакетов по данному каналу.

Поле **ImmAddress** содержит физический сетевой адрес станции, которой будут передаваться пакеты. Если станция-адресат находится в другой сети, в этом поле будет находиться адрес моста, через который пакет сможет дойти до адресата.

Поля **DestNetwork, DestNode, DestSocket** содержат компоненты полного сетевого адреса удаленной станции, с которой локальная станция работает по данному каналу, - номер сети, физический адрес станции в сети и номер сокета.

В поле **RetransmissionCount** находится максимальное значение количества повторных передач пакетов, по достижении которого SPX делает вывод о невозможности завершения передачи.

Поле **EstimatedRoundtripDelay** содержит время (в тиках таймера), в течение которого SPX ждет прихода подтверждения приема пакета от удаленной станции. По истечении этого времени SPX начинает выполнять повторную передачу пакета.

Поле **RetransmittedPackets** содержит количество выполненных повторных передач пакета.

Поле **SuppressedPackets** содержит количество отвергнутых пакетов. Пакеты могут быть отвергнуты, если они уже были приняты ранее или в настоящий момент нет свободных ECB для их приема.

3.4. Простая система "клиент–сервер" на базе SPX

Приведем простейший пример, демонстрирующий использование основных функций SPX. Этот пример сделан на базе предыдущего, в котором две программы - клиент и сервер - общались между собой с помощью протокола IPX.

После определения наличия драйвера IPX и получения адреса его API программа-сервер с помощью функции SPXCheckSPXInstallation() определяет присутствие драйвера протокола SPX.

Затем открывается сокет для протокола IPX, подготавливается ECB для приема пакета от клиента. Этот пакет будет передаваться с помощью протокола IPX и предназначен для определения адреса клиента. Аналогично предыдущему примеру программа-клиент посылает пакет в текущую сеть с номером 00000000 по адресу FFFFFFFFFFFFh, т. е. всем станциям текущей сети. После того, как программа-сервер примет этот пакет, она сохранит в области памяти ClientImmAddress непосредственный адрес станции, на которой работает программа-клиент.

После этого программа-сервер, пользуясь полученным непосредственным адресом, посылает клиенту ответный IPX-пакет, сообщая о том, что сервер принял пакет от клиента.

Далее программа-сервер открывает еще один сокет, который будет использоваться протоколом SPX. Напомним, что для работы с протоколами IPX и SPX необходимо выделять разные сокеты.

Открыв сокет, сервер подготавливает блок ECB для приема SPX-пакета от клиента. В поле непосредственного адреса копируется непосредственный адрес клиента, полученный после приема от него IPX-пакета. Запрос на создание канала выдает функция SPXListenForSequencedPacket().

Далее программа-сервер подготавливает в структуре ConnECB блок ECB для создания канала и вызывает функцию создания канала с принимающей стороны SPXListenForConnection().

После создания канала программа-сервер ожидает прихода SPX-пакета, проверяя в цикле содержимое поля InUse блока LsECB, распределенного ранее функцией SPXListenForSequencedPacket() для приема SPX-пакетов.

После прихода SPX-пакета сервер закрывает оба сокета и завершает свою работу.

```
// ================================================
// Листинг 12. Сервер SPX
//
// Файл spxserv.c
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
```

```
#include <stdlib.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "ipx.h"
#include "spx.h"
#define BUFFER_SIZE 512
void main(void) {
// Используем сокет 0x4568
     static unsigned IPXSocket = 0x4567;
     static unsigned SPXSocket = 0x4568;
// Этот ECB используется для приема пакетов и для их передачи.
     struct ECB RxECB;
     struct ECB ConnECB, LsECB;
// Заголовки принимаемых и передаваемых пакетов
     struct IPX_HEADER RxHeader, TxHeader;
     struct SPX_HEADER ConnHeader, LsHeader;
// Буферы для принимаемых и передаваемых пакетов
     unsigned char RxBuffer[BUFFER_SIZE];
     unsigned char TxBuffer[BUFFER_SIZE];
     struct SPXParams Params;
     unsigned char ClientImmAddress[6];
     printf("\n*Сервер SPX*, (C) Фролов А., 1993\n\n");
// Проверяем наличие драйвера IPX и определяем
// адрес точки входа его API
     if(ipx_init() != 0xff) {
          printf("IPX не загружен!\n"); exit(-1);
     }
     if( SPXCheckSPXInstallation(&Params) != 0xFF) {
          printf("SPX не загружен!\n"); exit(-1);
     }
// Открываем сокет, на котором мы будем принимать пакеты
     if(IPXOpenSocket(SHORT_LIVED, &IPXSocket)) {
          printf("Ошибка при открытии сокета IPX\n");
          exit(-1);
     };
// Подготавливаем ECB для приема пакета
     memset(&RxECB, 0, sizeof(RxECB));
     RxECB.Socket             = IntSwap(IPXSocket);
     RxECB.FragmentCnt        = 2;
     RxECB.Packet[0].Address  = &RxHeader;
     RxECB.Packet[0].Size     = sizeof(RxHeader);
     RxECB.Packet[1].Address  = RxBuffer;
```

```
    RxECB.Packet[1].Size    = BUFFER_SIZE;
    IPXListenForPacket(&RxECB);
    printf("Ожидание запроса от клиента\n");
    printf("Для отмены нажмите любую клавишу\n");
    while(RxECB.InUse) {
         IPXRelinquishControl();
         if(kbhit()) {
              getch();
              RxECB.CCode = 0xfe;
              break;
         }
    }
    if(RxECB.CCode == 0) {
         printf("Принят запрос от клиента '%s'\n", RxBuffer);
         printf("Для продолжения нажмите любую клавишу\n");
         getch();
         memcpy(ClientImmAddress, RxECB.ImmAddress,6);
// Подготавливаем ECB для передачи пакета
// Поле ImmAddress не заполняем, так как там уже находится адрес
// станции клиента. Это потому, что мы только что приняли от
// клиента пакет данных и при этом в ECB установился непосред-
// ственный адрес станции, которая отправила пакет
         RxECB.Socket            = IntSwap(IPXSocket);
         RxECB.FragmentCnt       = 2;
         RxECB.Packet[0].Address = &TxHeader;
         RxECB.Packet[0].Size    = sizeof(TxHeader);
         RxECB.Packet[1].Address = TxBuffer;
         RxECB.Packet[1].Size    = BUFFER_SIZE;
// Подготавливаем заголовок пакета
         TxHeader.PacketType = 4;
         memset(TxHeader.DestNetwork, 0, 4);
         memcpy(TxHeader.DestNode, RxECB.ImmAddress, 6);
         TxHeader.DestSocket = IntSwap(IPXSocket);
// Подготавливаем передаваемые данные
         strcpy(TxBuffer, "SPX SERVER *DEMO*");
// Передаем пакет обратно клиенту
         IPXSendPacket(&RxECB);
         printf("Связь с сервером установлена\n");
         printf("Создаем SPX-канал\n\n");
// Открываем сокет для работы с протоколом SPX
         if(IPXOpenSocket(SHORT_LIVED, &SPXSocket)) {
              printf("Ошибка при открытии сокета SPX\n");
              exit(-1);
         };
```

```
// Подготавливаем ECB для приема пакета
          memset(&LsECB, 0, sizeof(LsECB));
          LsECB.Socket             = IntSwap(SPXSocket);
          memcpy(LsECB.ImmAddress, ClientImmAddress,6);
          LsECB.FragmentCnt        = 2;
          LsECB.Packet[0].Address = &LsHeader;
          LsECB.Packet[0].Size     = sizeof(LsHeader);
          LsECB.Packet[1].Address = RxBuffer;
          LsECB.Packet[1].Size     = BUFFER_SIZE;
          SPXListenForSequencedPacket(&LsECB);
// Подготавливаем заголовок пакета
          ConnHeader.PacketType = 5;
          ConnHeader.TransportControl = 0;
          memset(ConnHeader.DestNetwork, 0, 4);
          memcpy(ConnHeader.DestNode, ClientImmAddress, 6);
          ConnHeader.DestSocket = IntSwap(SPXSocket);
          memset(&ConnECB, 0, sizeof(ConnECB));
          ConnECB.Socket = IntSwap(SPXSocket);
          ConnECB.FragmentCnt        = 1;
          ConnECB.Packet[0].Address = &ConnHeader;
          ConnECB.Packet[0].Size     = sizeof(ConnHeader);
// Ожидаем запрос на создание канала
          SPXListenForConnection(&ConnECB,0,0);
          while(ConnECB.InUse) {
               IPXRelinquishControl();
               if(kbhit()) {
                    getch();
                    ConnECB.CCode = 0xfe;
                    break;
               }
          }
          if(ConnECB.CCode == 0) {
               printf("Канал %04.4X создан\n",
                    (unsigned)ConnECB.ConnectionId);
          }
// Ожидаем прихода SPX-пакета от клиента
          while(LsECB.InUse) {
               IPXRelinquishControl();
               if(kbhit()) {
                    getch();
                    LsECB.CCode = 0xfe;
                    break;
               }
          }
          if(LsECB.CCode == 0) {
```

```
                printf("Пакет принят: '%s'\n", RxBuffer);
            }
        }
// Закрываем сокеты
        IPXCloseSocket(&IPXSocket);
        IPXCloseSocket(&SPXSocket);
        exit(0);
}
```

Программа-клиент после проверки наличия драйверов IPX и SPX открывает два сокета для использования с протоколами IPX и SPX. Затем подготавливается блок ECB для передачи "широковещательного" пакета по адресу FFFFFFFFFFFFh в текущую сеть с номером 000000. На этот пакет должна откликнуться программа-сервер, если она работает в текущей сети.

После передачи пакета программа-клиент ожидает прихода пакета от сервера. Затем она подготавливает блок ECB для приема SPX-пакета и ставит его в очередь на прием при помощи функции SPXListenForSequencedPacket().

Затем программа-клиент подготавливает блок ECB для создания канала с программой-сервером и вызывает функцию создания канала с передающей стороны SPXEstablishConnection().

После того, как канал будет создан, в область памяти ConnID копируется идентификатор канала для использования при приеме и передаче SPX-пакетов.

Далее программа-клиент подготавливает SPX-пакет и блок ECB для передачи программе-серверу и при помощи функции SPXSendSequencedPacket() передает пакет.

После передачи SPX-пакета программа-клиент закрывает оба сокета и завершает свою работу.

```
// ================================================
// Листинг 13. Клиент SPX
//
// Файл spxclien.c
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "ipx.h"
#include "spx.h"
// Максимальный размер буфера данных
#define BUFFER_SIZE 512
void main(void) {
```

```
// Будем работать с сокетом 0x4567
     static unsigned IPXSocket = 0x4567;
     static unsigned SPXSocket = 0x4568;
// ECB для приема и передачи пакетов
     struct ECB RxECB, TxECB;
     struct ECB ConnECB, LsECB, SndECB;
// Заголовки принимаемых и передаваемых пакетов
     struct IPX_HEADER RxHeader, TxHeader;
     struct SPX_HEADER ConnHeader, LsHeader, SndHeader;
// Буферы для принимаемых и передаваемых данных
     unsigned char RxBuffer[BUFFER_SIZE];
     unsigned char TxBuffer[BUFFER_SIZE];
     struct SPXParams Params;
     unsigned char ServerImmAddress[6];
     unsigned MyConnID, ConnID;
     unsigned rc;
     printf("\n*Клиент SPX*, (C) Фролов А., 1993\n\n");
// Проверяем наличие драйвера IPX и определяем
// адрес точки входа его API
     if(ipx_init() != 0xff) {
          printf("IPX не загружен!\n"); exit(-1);
     }
     if( SPXCheckSPXInstallation(&Params) != 0xFF) {
          printf("SPX не загружен!\n"); exit(-1);
     }
// Открываем сокет, на котором мы будем
// принимать и передавать пакеты
     if(IPXOpenSocket(SHORT_LIVED, &IPXSocket)) {
          printf("Ошибка при открытии сокета\n");
          exit(-1);
     };
// Отхрываем сокет для протокола SPX
     if(IPXOpenSocket(SHORT_LIVED, &SPXSocket)) {
          printf("Ошибка при открытии сокета SPX\n");
          exit(-1);
     };

// Подготавливаем ECB для передачи пакета
     memset(&TxECB, 0, sizeof(TxECB));
     TxECB.Socket             = IntSwap(IPXSocket);
     TxECB.FragmentCnt        = 2;
     TxECB.Packet[0].Address  = &TxHeader;
     TxECB.Packet[0].Size     = sizeof(TxHeader);
```

```
    TxECB.Packet[1].Address = TxBuffer;
    TxECB.Packet[1].Size    = BUFFER_SIZE;
// Пакет предиазначен всем станциям данной сети
    memset(TxECB.ImmAddress, 0xff, 6);
// Подготавливаем заголовок пакета
    TxHeader.PacketType = 4;
    memset(TxHeader.DestNetwork, 0, 4);
    memset(TxBeader.DestNode, 0xff, 6);
    TxHeader.DestSocket = IntSwap(IPXSocket);
// Записываем передаваемые даиные
    strcpy(TxBuffer, "CLIENT *DEMO*");
// Передаем пакет всем станциям в даниой сети
    IPXSendPacket(&TxECB);
// Подготавливаем ECB для приема пакета от сервера
    memset(&RxECB, 0, sizeof(RxECB));
    RxECB.Socket            = IntSwap(IPXSocket);
    RxECB.FragmentCnt       = 2;
    RxECB.Packet[0].Address = &RxHeader;
    RxECB.Packet[0].Size    = sizeof(RxHeader);
    RxECB.Packet[1].Address = RxBuffer;
    RxECB.Packet[1].Size    = BUFFER_SIZE;
    IPXListenForPacket(&RxECB);
    printf("Ожидание ответа от сервера\n");
    printf("Для отмеиы иажмите любую клавишу\n");
// Ожидаем прихода ответа от сервера
    while(RxECB.InUse) {
        IPXRelinquishControl();
        if(kbhit()) {
            getch();
            RxECB.CCode = 0xfe;
            break;
        }
    }
    if(RxECB.CCode == 0) {
        printf("Принят ответ от сервера '%s'\n", RxBuffer);
    }
// Копируем сетевой адрес сервера
    memcpy(ServerImmAddress, RxECB.ImmAddress, 6);
// Подготавливаем ECB для приема пакета
    memset(&LsECB, 0, sizeof(LsECB));
    LsECB.Socket            = IntSwap(SPXSocket);
    memcpy(LsECB.ImmAddress, ServerImmAddress,6);
    LsECB.FragmentCnt       = 2;
```

```
    LsECB.Packet[0].Address = &LsHeader;
    LsECB.Packet[0].Size    = sizeof(LsHeader);
    LsECB.Packet[1].Address = RxBuffer;
    LsECB.Packet[1].Size    = BUFFER_SIZE;
    SPXListenForSequencedPacket(&LsECB);
// Подготавливаем заголовок пакета
    ConnHeader.PacketType = 5;
    ConnHeader.TransportControl = 0;
    memset(ConnHeader.DestNetwork, 0, 4);
    memcpy(ConnHeader.DestNode, ServerImmAddress, 6);
    ConnHeader.DestSocket = IntSwap(SPXSocket);
    memset(&ConnECB, 0, sizeof(ConnECB));
    ConnECB.Socket = IntSwap(SPXSocket);
    ConnECB.FragmentCnt       = 1;
    ConnECB.Packet[0].Address = &ConnHeader;
    ConnECB.Packet[0].Size    = sizeof(ConnHeader);
// Устанавливаем SPX-канал с сервером
    rc = SPXEstablishConnection(&ConnECB, &MyConnID, 0, 0);
    printf("Ожидание SPX-соединения с сервером\n");
    printf("Для отмены нажмите любую клавишу\n");
    if(rc == 0) {
        while(ConnECB.InUse) {
            IPXRelinquishControl();
            if(kbhit()) {
                getch();
                ConnECB.CCode = 0xfe;
                break;
            }
        }
    }
// Копируем идентификатор канала для передачи пакета в сервер
    memcpy(&ConnID, &(ConnHeader.SourceConnID), 2);
    printf("Канал с сервером установлен, ConnID=%d\n",
        IntSwap(ConnID));
// Подготавливаем ECB для передачи SPX-пакета
    memset(&SndECB, 0, sizeof(SndECB));
    SndECB.Socket            = IntSwap(SPXSocket);
    SndECB.FragmentCnt       = 2;
    SndECB.Packet[0].Address = &SndHeader;
    SndECB.Packet[0].Size    = sizeof(SndHeader);
    SndECB.Packet[1].Address = TxBuffer;
    SndECB.Packet[1].Size    = BUFFER_SIZE;
    memcpy(SndECB.ImmAddress, ServerImmAddress, 6);
// Подготавливаем заголовок пакета
```

```
    SndHeader.PacketType = 5;
    memset(SndHeader.DestNetwork, 0, 4);
    memcpy(SndHeader.DestNode, ServerImmAddress, 6);
    SndHeader.DestSocket = IntSwap(SPXSocket);
    SndHeader.TransportControl = 0;
    SndHeader.DataStreamType = 1;
// Записываем передаваемые данные
    strcpy(TxBuffer, "SPX/CLIENT *DEMO*");
// Передаем SPX-пакет
    SPXSendSequencedPacket(&SndECB, ConnID);
// Закрываем сокеты
    IPXCloseSocket(&IPXSocket);
    IPXCloseSocket(&SPXSocket);
    exit(0);
}
```

В файле spx.c определены функции для работы с протоколом SPX (листинг 14):

```
// ================================================
// Листинг 14. Фуикции SPX.
//
// Файл spx.c
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include "ipx.h"
#include "spx.h"
/**
* .Name    SPXCheckSPXInstallation
*
* .Title   Проверить присутствие протокола SPX
*
* .Descr   Функция проверяет, загружен ли драйвер SPX
*          и возвращает его параметры.
*
* .Params  struct *SPXParams - указатель на структуру,
*          в которую будут записаны параметры SPX.
*
* .Return  FFh - протокол SPX загружен
*          00h - протокол SPX не загружен
**/
int SPXCheckSPXInstallation(struct SPXParams *Params) {
    struct IPXSPX_REGS iregs;
```

```
    iregs.bx = SPX_CMD_INSTALL_CHECK;
    iregs.ax = 0;
    ipxspx_entry( (void far *)&iregs );
    Params->SPXVersion = iregs.bx;
    Params->SPXMaxConnections = iregs.cx;
    Params->SPXAvailableConnCount = iregs.dx;
    return(iregs.ax & 0xFF);
}
/**
* .Name    SPXListenForConnection
*
* .Title   Ожидание соединения с клиентом
*
* .Descr   Функция выдает запрос на соединеиие
*          с клиеитом, который должен для выполиения
*          соединения вызвать фуикцню SPXEstablishConnection().
*
* .Params  struct ECB *ConnECB - указатель на ECB,
*                    заполиенное для устаиовлеиия соеднненя.
*          unsigned char RetryCount   - счетчик повторов;
*          unsigned char WatchdogFlag - проверка связн.
*
* .Return  Ничего.
**/
void SPXListenForConnection(struct ECB *ConnECB,
    unsigned char RetryCount, unsigned char WatchdogFlag) {
    struct IPXSPX_REGS iregs;
    iregs.bx = SPX_CMD_LISTEN_FOR_CONNECTION;
    iregs.ax = RetryCount |
        ((unsigned)(WatchdogFlag << 8) & 0xff00);
    iregs.es = FP_SEG((void far*)ConnECB);
    iregs.si = FP_OFF((void far*)ConnECB);
    ipxspx_entry( (void far *)&iregs );
}
/**
* .Name    SPXEstablishConnection
*
* .Title   Установление соединения с клиентом
*
* .Descr   Функция устанавливает соединение
*          с клиентом, который должен для выполиения
*          соеднненя вызвать функцню SPXListenForConnection().
*
* .Params  struct ECB *ConnECB - указатель на ECB,
*                    заполнениый для установления соединиения.
*          unsigned char RetryCount   - счетчик повторов;
```

```
*          unsigned char WatchdogFlag - проверка связи.
*
* .Return  Ничего.
**/
int SPXEstablishConnection(struct ECB *ConnECB, unsigned *ConnID,
     unsigned char RetryCount, unsigned char WatchdogFlag) {
     struct IPXSPX_REGS iregs;
     iregs.bx = SPX_CMD_ESTABLISH_CONNECTION;
     iregs.ax = RetryCount |
          ((unsigned)(WatchdogFlag << 8) & 0xff00);
     iregs.es = FP_SEG((void far*)ConnECB);
     iregs.si = FP_OFF((void far*)ConnECB);
     ipxspx_entry( (void far *)&iregs );
     *ConnID = iregs.dx;
     return(iregs.ax & 0xff);
}
/**
* .Name    SPXListenForSequencedPacket
*
* .Title   Прием пакета SPX
*
* .Descr   Функция выдает запрос на прием пакета SPX.
*
* .Params  struct ECB *LsECB - указатель на ECB,
*                    заполненный для приема SPX-пакета.
*
* .Return  Ничего.
**/
void SPXListenForSequencedPacket(struct ECB *LsECB) {
     struct IPXSPX_REGS iregs;
     iregs.bx = SPX_CMD_LISTEN_FOR_SEQUENCED_PACKET;
     iregs.es = FP_SEG((void far*)LsECB);
     iregs.si = FP_OFF((void far*)LsECB);
     ipxspx_entry( (void far *)&iregs );
}
/**
* .Name    SPXSendSequencedPacket
*
* .Title   Передача пакета SPX
*
* .Descr   Функция выдает запрос на передачу пакета SPX.
*
* .Params  struct ECB *TxECB - указатель на ECB,
*                    заполненный для передачи SPX-пакета.
*
```

```
* .Return  Ничего.
**/
void SPXSendSequencedPacket(struct ECB *TxECB,unsigned ConnID) {
      struct IPXSPX_REGS iregs;
      iregs.bx = SPX_CMD_SEND_SEQUENCED_PACKET;
      iregs.es = FP_SEG((void far*)TxECB);
      iregs.si = FP_OFF((void far*)TxECB);
      iregs.dx = ConnID;
      ipxspx_entry( (void far *)&iregs );
}
```

В файле spx.h (листинг 15) определены константы, структуры данных и прототипы функций для работы с протоколом SPX:

```
// ================================================
// Листинг 15. Include-файл для работы с SPX
// Файл spx.h
//
// (C) A. Frolov, 1992
// ================================================
#include <dos.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
// ----------------------
// Команды интерфейса SPX
// ----------------------
#define SPX_CMD_INSTALL_CHECK              0x10
#define SPX_CMD_ESTABLISH_CONNECTION       0x11
#define SPX_CMD_LISTEN_FOR_CONNECTION      0x12
#define SPX_CMD_TERMINATE_CONNECTION       0x13
#define SPX_CMD_ABORT_CONNECTION           0x14
#define SPX_CMD_GET_CONNECTION_STATUS      0x15
#define SPX_CMD_SEND_SEQUENCED_PACKET      0x16
#define SPX_CMD_LISTEN_FOR_SEQUENCED_PACKET     0x17
struct SPXParams {
      unsigned SPXVersion;
      unsigned SPXMaxConnections;
      unsigned SPXAvailableConnCount;
};
// ============================================================
// Заголовок пакета SPX
// ============================================================
struct SPX_HEADER {
           unsigned int    Checksum;
           unsigned int    Length;
```

```
            unsigned char   TransportControl;
            unsigned char   PacketType;
            unsigned char   DestNetwork[4];
            unsigned char   DestNode[6];
            unsigned int    DestSocket;
            unsigned char   SourceNetwork[4];
            unsigned char   SourceNode[6];
            unsigned int    SourceSocket;
// ------------Специфическая для SPX часть ---------
            unsigned char  ConnControl;
            unsigned char  DataStreamType;
            unsigned char  SourceConnID[2];
            unsigned char  DestConnID[2];
            unsigned char  SequenceNumber[2];
            unsigned char  AckNumber[2];
            unsigned char  AllocationNumber[2];
};
int SPXCheckSPXInstallation(struct SPXParams *Params);
void SPXListenForConnection(struct ECB *ConnECB,
     unsigned char RetryCount, unsigned char WatchdogFlag);
int SPXEstablishConnection(struct ECB *ConnECB, unsigned *ConnID,
     unsigned char RetryCount, unsigned char WatchdogFlag);
void SPXListenForSequencedPacket(struct ECB *LsECB);
void SPXSendSequencedPacket(struct ECB *TxECB, unsigned MyConnID);
```

3.5. Настройка параметров SPX

В разделе, посвященном настройке параметров драйвера IPX, мы говорили о том, что при запуске программы ipxodi.com можно указывать параметры. Если указывается параметр "d", на рабочей станции не загружается диагностический сервис. Если же указывается параметр "a", в память не загружаются драйвер протокола SPX и диагностический сервис.

В документации на вашу программу следует указать о том, какие параметры можно использовать при загрузке ipxodi.com. В частности, если ваша программа использует протокол SPX, параметр "a" задавать нельзя.

Для изменения режима работы драйвера SPX в первых строках файла net.cfg, расположенного в каталоге C:\NET (см. предыдущий том "Библиотеки системного программиста"), можно указывать параметры:

SPX ABORT TIMEOUT — Время в тиках системного таймера, в течение которого драйвер SPX будет ожидать прихода ответа от партнера по каналу, прежде чем будет сделан вывод о невозможности работы с каналом. После истечения этого времени канал будет закрыт.
По умолчанию драйвер ждет 540 тиков, что соответствует примерно 30 с.

SPX CONNECTIONS	Параметр определяет максимальное количество каналов, которые могут быть созданы на рабочей станции. По умолчанию можно создавать максимально 15 каналов.
SPX LISTEN TIMEOUT	Параметр задает время, в течение которого драйвер SPX будет ждать прихода пакета от партнера. Если за это время пакет не придет, драйвер будет посылать пакеты для проверки работоспособности канала. По умолчанию это время равно 108 тикам, что составляет примерно 6 с.
SPX VERIFY TIMEOUT	Этот параметр задает период времени, с которым драйвер SPX передает пакеты для проверки работоспособности канала связи. По умолчанию проверочные пакеты передаются с интервалом 54 тика (примерно 3 с).

Например, для увеличения числа доступных каналов до 25 добавьте в начало файла net.cfg строку:

```
SPX CONNECTIONS=25
```

Глава 4

ПРОТОКОЛ NETBIOS

Последний протокол, который мы рассмотрим в нашей книге, - протокол NETBIOS (Network Basic Input/Output System - базовая сетевая система ввода/вывода), разработанный IBM. Этот протокол работает на трех уровнях семиуровневой модели OSI: сетевом уровне, транспортном уровне и на уровне каналов связи. Уровень каналов связи обеспечивает механизм обмена сообщениями между программами, работающими на станциях в рамках канала связи или сессии. NETBIOS может обеспечить интерфейс более высокого уровня, чем протоколы IPX и SPX.

Протокол NETBIOS поддерживается в сетях IBM (IBM PC LAN), Novell NetWare, Microsoft Windows for Workgroups и в других сетях. К сожалению, нет единого стандарта на протокол NETBIOS, поэтому в сетевом программном обеспечении разных фирм используются разные интерфейсы для вызова команд NETBIOS.

С нашей точки зрения, наибольший интерес представляет применение NETBIOS в сетях Novell NetWare и Microsoft Windows for Workgroups. Мы рассмотрим основные возможности NETBIOS, связанные с передачей данных между рабочими станциями в пределах одного логического сегмента сети.

Для работы с протоколом NETBIOS в сетях Novell NetWare необходимо запустить специальный эмулятор NETBIOS - программу **netbios.exe**, входящую в комплект поставки Novell NetWare. Эта программа эмулирует протокол NETBIOS с использованием уже знакомых нам протоколов IPX/SPX.

Использовать NETBIOS проще, чем IPX или SPX. Однако, так как в среде Novell NetWare нужен специальный эмулятор NETBIOS, эффективность работы программы может снизиться. Кроме того, для эмулятора нужна дополнительная память, так как он реализован в виде резидентной программы.

4.1. Адресация станций и программ

Как вы помните, для идентификации рабочей станции протоколы IPX и SPX используют номер сети, адрес станции в сети и сокет. Адрес станции определяется на аппаратном уровне и представляет собой число длиной 6 байт. Номер сети занимает 4 байта. Сокеты выделяются динамически драйвером протокола IPX или могут быть получены в Novell.

Протокол NETBIOS использует другой механизм адресации станций и программ. Для адресации станции используются имена размером 16 байт. Каждая станция имеет одно *постоянное имя* (permanent name), которое образуется из аппаратного адреса добавлением к нему слева десяти нулевых байт. Кроме постоянного имени протокол NETBIOS позволяет добавлять (и удалять) *обычные имена и групповые имена*. Обычные имена служат для идентификации рабочей станции, групповые могут служить для посылки пакетов одновременно нескольким станциям в сети. Постоянное имя удалить нельзя, так как оно полностью определяется аппаратным обеспечением станции.

При добавленнн обычного имени протокол NETBIOS опрашивает всю сеть для проверкн уникальности нмени. Групповое нмя может быть одинаковое на иескольknх станциях, поэтому прн добавлении группового имени опрос сети не выполияется.

После добавления нового именн этому имени присваивается так называемый иомер имени (name number), который используется для передачи данных по сети.

Сравнивая методы адресации, используемые протоколами IPX/SPX и NETBIOS, нетрудно заметить, что метод адресации протокола NETBIOS более удобен. Вы можете адресовать данные не только одной станцни (как в IPX н SPX) илн всем станциям сразу (как в IPX), но и группам станций, имеющим одинаковое групповое имя. Это может быть удобно, если в сети работают несколько групп пользователей, которые интенсивно обмениваются данными между собой.

Другим преимуществом схемы адресации протокола NETBIOS перед схемой адресацни протоколов IPX/SPX можно считать отсутствие необходимости получать в фирме Novell свой собственный номер сокета для идентификации вашего программного обеспечения. Вы можете придумать свое собственное уникальное групповое имя, включающее, например, название программы н вашей фирмы, и использовать его для работы по схеме клиент–сервер.

4.2. Работа с протоколом NETBIOS

Протокол NETBIOS предоставляет программам интерфейс для передачи данных на уровне датаграмм н на уровне каналов связи. Для вызова NETBIOS программа должна создать в памяти управляющий блок, который называется NCB (Network Control Block - сетевой управляющий блок). Адрес заполненного блока NCB передается прерыванию INT 5Ch, в рамках которого и реализован нитерфейс протокола NETBIOS. Есть также альтернативный интерфейс, реалнзованный в рамках прерывания INT 2Ah, который поддерживается эмулятором NETBIOS, разработанным фирмой Novell, а также операцнонной системой Windows for Workgroups версии 3.1.

4.2.1. Проверка присутствия NETBIOS

Первое, что должна сделать программа, желающая воспользоваться протоколом NETBIOS, - проверить наличие в системе интерфейса NETBIOS.

Ниже приведена программа, которая определяет, установлен ли драйвер NETBIOS.

С помощью функцин getvect() программа получает указатель на обработчик прерывания INT 5Ch. Это прерывание используется для вызова NETBIOS. Если сегментная компонента адреса равна нулю нли F000h, обработчик прерывания не установлен или установлена заглушка, расположенная в BIOS. В этом случае программа считает, что NETBIOS отсутствует.

```
// ================================================
// Листинг 16. Проверка присутствия NETBIOS
//
// Файл nbver.cpp
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
void main(void) {
     void interrupt ( *int5C)(...);
     printf("Check if NETBIOS is installed\n");
     int5C = getvect(0x5c);
     if(FP_SEG(int5C) == 0x0000 || FP_SEG(int5C) == 0xF000) {
          printf("NETBIOS NOT installed.\n");
     }
     else printf("NETBIOS is installed!\n");
}
```

Другой способ проверки наличия интерфейса NETBIOS заключается в вызове прерывания INT 2Ah. Загрузите в регистр AH нулевое значение и вызовите прерывание INT 2Ah. Если после возврата из прерывания в регистре AH попрежнему находится нуль, драйвер NETBIOS не установлен.

Данный способ проверки будет работать на виртуальной машине DOS, запущенной в среде Windows for Workgroups версии 3.1 (если Windows работает в расширенном режиме).

Приведенная ниже программа определяет присутствие NETBIOS с помощью вызова прерывания INT 2Ah (листинг 17):

```
// ================================================
// Листинг 17. Проверка присутствия NETBIOS
//             с использованием интерфейса INT 2Ah
//
// Файл 2atest.cpp
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
```

```
void main(void) {
     union REGS regs;
     printf("Check if NETBIOS is installed\n");
     regs.h.ah = 0;
     int86(0x2a, &regs, &regs);
     if(regs.h.ah  == 0) {
          printf("NETBIOS NOT installed.\n");
     }
     else printf("NETBIOS is installed!\n");
}
```

Прерывание INT 2Ah используется в сетях фирм Microsoft и Lantastic. Эмулятор протокола NETBIOS, поставляющийся фирмой Novell вместе с операционной системой Novell NetWare, поддерживает как интерфейс прерывания INT 5Ch, так и интерфейс INT 2Ah.

4.2.2. Вызов команд протокола NETBIOS

Интерфейс протокола NETBIOS реализован в рамках прерывания INT 5Ch или INT 2Ah и очень прост.

Для вызова команд протокола NETBIOS вам достаточно подготовить блок NCB, загрузить его дальний адрес в регистры ES:BX и вызвать прерывание INT 5Ch.

Приведем формат вызова функции прерывания INT 2Ah, предназначенной для выполнения команд NETBIOS:

На входе: AH = 04h.
AL = Признак автоматического повтора команды при получении кодов ошибки 09h (недостаточно ресурсов), 12h (создание канала отвергнуто, так как на другом конце не выдана команда NB_Listen), 21h (занят интерфейс):
00h - повторять команду;
FFh - не повторять команду.
ES:BX = Адрес заполненного блока NCB.

На выходе: AH = Код завершения:
00h - команда выполнилась без ошибок;
01h - при выполнении команды были ошибки.
AL = Если содержимое регистра AH после возврата из прерывания не равно нулю, регистр AL содержит код ошибки

Вы можете использовать описанный выше вызов прерывания INT 2A для вызова NETBIOS из программы, работающей на виртуальной машине Windows for Workgroups версии 3.1.

Для вызова команд протокола NETBIOS из программы, составленной на языке Си, вы можете воспользоваться обычными средствами вызова программных прерываний, такими, как функция int86x().

4.2.3. Формат блока NCB

Приведем формат блока NCB:

```
struct _NCB {
     unsigned char Cmd;
     unsigned char CCode;
     unsigned char LocalSessionNumber;
     unsigned char NetworkNameNumber;
     void far *Buffer;
     unsigned Size;
     char CallName[16];
     char OurName[16];
     unsigned char ReceiveTimeout;
     unsigned char SendTimeout;
     void interrupt (*PostRoutine)(void);
     unsigned char AdapterNumber;
     unsigned char FinalCCode;
     unsigned char Reserved[14];
};
```

Поле **Cmd** содержит код команды, которую необходимо выполнить. Существуют команды для работы с именами станций, для передачи и приема датаграмм, для работы на уровне каналов и некоторые другие. Мы рассмотрим команды NETBIOS в следующем разделе.

Поле **CCode** содержит код ошибки, возвращаемый NETBIOS до выполнения команды. Если, например, программа затребовала выполнение неправильной команды или задала для команды неправильные параметры, NETBIOS не будет выполнять такую команду и установит в поле CCode соответствующий код ошибки. Если же в этом поле после вызова NETBIOS находится нулевое значение, это еще не означает, что команда выполнилась правильно, однако она начала выполняться.

Поле **LocalSessionNumber** содержит номер канала, установленного с другой программой. Это поле необходимо заполнять при выдаче команд передачи данных через каналы.

Поле **NetworkNameNumber** содержит номер имени, который присваивается при добавлении обычного или группового имени. Это поле должно быть заполнено при приеме датаграмм.

Поле **Buffer** представляет собой дальний указатель в формате [сегмент:смещение] на буфер, который должен содержать данные перед выполнением передачи или на буфер, который будет использован для приема данных.

Размер буфера, используемого для приема или передачи данных, определяется содержимым поля **Size**.

Поле **CallName** заполняется именем станции, с которой ваша станция желает установить канал для передачи данных.

Поле **OurName** должно содержать имя вашей программы, под которым она будет принимать данные. В качестве этого имени может выступать обычное, групповое или постоянное имя.

Поля **ReceiveTimeout** и **SendTimeout** содержат интервал времени (измеряемый в 1/2 с), в течение которого ожидается завершение соответственно команд приема и передачи.

Поле **PostRoutine** - указатель на программу, которая получает управление после завершения команды. Эта программа (POST-программа) аналогична программе ESR протоколов IPX/SPX и вызывается только в том случае, если в поле PostRoutine был указан адрес программы. Если же это поле заполнить нулями, никакая программа вызываться не будет.

Поле **AdapterNumber** используется, если в станции установлено несколько сетевых адаптеров (в сетях Ethernet этого обычно не бывает). В этом поле указывается номер адаптера, для которого предназначена команда. Первый адаптер имеет номер 0, второй - 1.

Поле **FinalCCode** содержит во время выполнения команды значение 0xFF. После завершения выполнения команды в это поле записывается код ошибки, который относится к выполнению команды в целом (в отличие от кода в поле CCode). Если ваша программа не задала адрес для программы в поле PostRoutine, она должна опрашивать в цикле содержимое этого поля, ожидая, пока в нем не появится значение, отличное от 0xFF.

Поле **Reserved** зарезервировано для использования протоколом NETBIOS, ваша программа не должна изменять его содержимое.

4.2.4. POST-программа

POST-программа является программой обработки прерывания. Она получает управление в состоянии с запрещенными прерываниями. Регистры ES:BX содержат адрес блока NCB, который использовался при выполнении команды. В регистр AL записано значение из поля FinalCCode блока NCB.

Учтите, что, как и из любой другой программы обработки прерывания, из POST-программы не следует вызывать функции MS-DOS.

Требования к POST-программе во многом такие же, как и к ESR-программе, используемой протоколами IPX и SPX. Она должна позаботиться о сохранении изменяемых регистров, установить регистр DS на сегмент данных программы (для обеспечения доступа к переменным). POST-программа должна работать как можно быстрее. Лучше всего если она будет использоваться только для установки флага, сигнализирующего основной программе о завершении выполнения команды.

Перед завершением своей работы POST-программа должна восстановить содержимое всех регистров и выполнить команду возврата из прерывания **IRET**.

Если вы составляете POST-программу на языке программнрования Си, вы можете воспользоваться ключевым словом interrupt:

```
void interrupt NETBIOS_Post_Routine(void);
```

Лучше всего составить POST-программу на языке ассемблера, например, так:

```
.286
.MODEL SMALL
.DATA
     _completed_ncb_ptr dd 0
.CODE
     PUBLIC     _netbios_post
     PUBLIC     _completed_ncb_ptr
_netbios_post PROC FAR
          push ax
          push ds  .
          push es
          push si
          mov  ax, DGROUP
          mov  ds, ax
          mov  word ptr _completed_ncb_ptr+2, es
          mov  word ptr _completed_ncb_ptr, si
          pop  si
          pop  es
          pop  ds
          pop  ax
          iret
_netbios_post ENDP
end
```

4.3. Команды NETBIOS

Перед выполнением команды ее код должен быть записан в поле Cmd блока NCB. Каждая команда NETBIOS реализована в двух вариантах - с ожиданием и без ожидания.

Если вашей программе нечего делать до тех пор, пока выполнение команды NETBIOS не будет полностью завершено, вы можете выбрать вариант с ожиданием. В этом случае после вызова NETBIOS программа вновь получит управленне только после завершения выполнения команды. Прн нспользованин протоколов IPX/SPX ваша программа должна была сама дожидаться выполнения вызванной функции, выполняя в цикле опрос поля InUse блока ECB. Однако учтите, что, если по каким-либо причинам выполненне команды не может быть завершено, ваша программа "зависнет".

Вариант без ожидания похож на вариант использования функций IPX/SPX с программой ESR, вызываемой после завершения операции. Программа, вызвавшая команду NETBIOS без ожидания, получает управление немедленно. Команда будет выполняться в фоновом режиме параллельно с работой вызвавшей ее программы. После того как выполнение команды будет завершено, управление будет передано функции, адрес которой необходимо указать в поле PostRoutine блока NCB. Можно также дожидаться окончания выполнения команды, опрашивая в цикле поле FinalCCode, которое будет содержать значение 0xFF до тех пор, пока команда не будет выполнена.

Все команды NETBIOS можно разделить на несколько групп:

- для работы с именами;
- для приема и передачи датаграмм;
- для работы с каналами;
- для приема и передачи данных через каналы;
- другие команды.

Так как большинство команд NETBIOS реализованы в двух вариантах (с ожиданием и без ожидания), для обозначения варианта с ожиданием мы будем в названии команды после префикса NB_ (от слова NETBIOS) добавлять букву W (от слова wait - ожидание). Например, команда NB_AddName выполняется без ожидания, а команда NB_WAddName - с ожиданием.

У команд без ожидания старший бит кода команды установлен в единицу.

4.3.1. Работа с именами

В этой группе есть команды, позволяющие добавлять обычное или групповое имя, удалять имя. Все эти команды могут работать в двух вариантах - с ожиданием и без ожидания.

NB_WAddName (0x30)

Команда добавляет указанное в поле OurName имя в таблицу имен, расположенную на рабочей станции. Имя должно быть уникальным в сети. Оно не может использоваться на других станциях ни как обычное, ни как групповое.

Если длина имени меньше 16 байт, оно должно быть дополнено справа символами пробела. Можно закрыть имя двоичным нулем для совместимости со строками языка Си. Нуль должен находиться в последней позиции имени.

После успешного выполнения команды NETBIOS присваивает имени номер и возвращает его в поле LocalSessionNumber блока NCB. Номер имени может потребоваться вам для работы с датаграммами.

В процессе добавления имени NETBIOS посылает по сети запрос. Если такое имя уже используется на какой-либо станции, эта станция пришлет ответ. В этом случае команда завершиться с ошибкой и имя добавлено не будет.

Процедура добавления имени занимает достаточно много времени. Это связано с необходимостью выполнить опрос всех станций сети. Поэтому вы должны добавлять имена один раз в самом начале работы программы.

Поля NCB на входе	*Содержимое*
Cmd	0x30
OurName	Добавляемое имя
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
NetworkNameNumber	Присвоенный номер имени
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x09	Нет доступных ресурсов
0x0D	Указанное команде имя уже используется на этой станции
0x0E	Переполнение таблицы имен
0x15	Неправильное имя
0x16	Имя уже используется на одной из рабочих станций в сети
0x19	Конфликт имен (внутренняя ошибка NETBIOS)
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

NB_AddName (0xB0)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0xB0.

NB_WAddGroupName (0x36)

Команда добавляет указанное в поле OurName групповое имя в таблицу имен, расположенную на рабочей станции.

Имя не должно использоваться другими станциями в сети как обычное. Однако несколько станций могут использовать одно и тоже имя как групповое.

Если длина имени меньше 16 байт, оно должно быть дополнено справа символами пробела. Можно закрыть имя двоичным нулем для совместимости со строками языка Си. Нуль должен находиться в последней позиции имени.

После успешного выполнения команды NETBIOS присваивает имени номер и возвращает его в поле LocalSessionNumber блока NCB. Номер имени нужен для работы с датаграммами.

Поля NCB на входе	*Содержимое*
Cmd	0x36
OurName	Добавляемое групповое имя
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
NetworkNameNumber	Присвоенный номер имени
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильиый код команды
0x09	Нет доступиых ресурсов
0x0D	Указаниое команде имя уже используется иа этой станцин
0x0E	Переполиеиие таблицы имеи
0x15	Неправильиое нмя
0x16	Имя уже испо́льзуется на одной нз рабочих станций в сети
0x19	Коифликт имеи (внутреиняя ошибка NETBIOS)
0x21	Интерфейс занят
0x22	Выдано слишком миого команд
0x23	Неправильное значеиие в поле иомера адаптера AdapterNumber
0x40 - 0x4F	Необычиое состояиие сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечеиия

NB_AddGroupName (0xB6)

Команда аналогичиа предыдущей, за исключеинем того, что оиа выпо́лняется без ожидания и в поле Cmd необходимо записать зиачеине 0xB6.

NB_WDeleteName (0x31)

Команда удаляет имя из таблицы имен рабочей станции, если оио не нспользуется каким-либо каналом. Еслн же имя нспользуется каиалом, то оио помечается как назиачениое для удаления н удаляется после закрытия канала.

Если вы попытаетесь удалить имя, которое используется каналом, команда завершится с кодом ошибки 0xF. В этом случае перед удаленнем имени иеобходимо закрыть канал (см. дальше описание команд для работы с каналами). Как только канал будет закрыт, связанное с ним имя будет автоматическн удалено, еслн перед закрытием канала выполнялась попытка удалить имя.

Еслн имя нспользуется нескольким каналами, его можно удалить только после закрытия всех связаниых с ннм каналов.

Поля NCB на входе	*Содержимое*
Cmd	0x31
OurName	Удаляемое имя
PostRoutine	Дальний указатель иа POST-программу или нулевое значение, еслн POST-программа не нспользуется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточный код завершения команды
FinalCCode	Окончательиый код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x21	Интерфейс занят
0x22	Выдано слншком много команд
0x23	Неправильиое значенне в поле иомера адаптера AdapterNumber
0x40 - 0x4F	Необычиое состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечеиия

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x0F	Имя используется каналом. Немедлениое удаление имени невозможно, однако оио отмечено как иазначениое для удаления н будет удалено после закрытия канала
0x15	Неправильиое имя
0x21	Интерфейс занят
0x22	Выдано слишком миого команд
0x23	Неправильиое зиачеиие в поле иомера адаптера AdapterNumber
0x40 - 0x4F	Необычиое состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

NB_DeleteName (0xB1)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd иеобходимо записать зиачеине 0xB1.

4.3.2. Прием и передача датаграмм

С помощью команд приема и передачи датаграмм вы можете передавать и прииимать пакеты без подтверждеиия, аналогичио тому, как это выполняет протокол IPX.

Есть команды для передачи и приема датаграмм по обычиому или групповому имени, а также для передачи и приема датаграмм, адресованных одновременио всем станциям в сети.

В отличие от протокола IPX протокол NETBIOS использует разные команды для приема обычных датаграмм и датаграмм, адресованных всем станциям в сети.

Протокол IPX ие позволяет вам передавать пакет группе станций в сети. Вы можете передать пакет либо какой-либо одной станции, либо всем станциям сразу. С помощью NETBIOS вы можете организовать передачу данных так, что пакеты будут приниматься только одной группой станций в сети по ее групповому имени.

Длнна сообщений, передаваемых при помощи команд данной группы, ограничеиа 512 байтами. Через каналы вы можете передавать блоки данных существенно большего размера.

NB_WSendDatagram (0x20)

Команда предиазиачеиа для передачи блока даниых размером от 1 до 512 байт в виде датаграммы (без подтверждеиия приема). Датаграмма может быть послана на обычиое или групповое имя.

Для передачи датаграммы вам ие надо создавать канал с принимающей станцией.

Поля NCB на входе	*Содержимое*
Cmd	0x20
NetworkNameNumber	Номер, присвоеииый при добавлении имени
CallName	Имя станции, которой передаются даниые
Buffer	Адрес буфера, содержащего передаваемые данные
Size	Размер буфера
PostRoutine	Дальний указатель иа POST-программу или нулевое зиачеиие, если POST-программа ие используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточиый код завершеиия команды
FinalCCode	Окончательный код завершеиия команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильиый код команды
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильиое зиачеиие в поле иомера адаптера AdapterNumber

0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x01	Неправильная длина буфера
0x03	Неправильный код команды
0x13	Неправильный номер имеин
0x19	Конфликт имен (внутренняя ошибка NETBIOS)
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера илн другого сетевого аппаратного обеспечения

NB_SendDatagram (0xA0)

Команда аналогична предыдущей, за исключеннем того, что она выполняется без ожидания и в поле Cmd иеобходнмо записать значеиие 0xA0.

NB_WSendBroadcastDatagram.(0x22)

Команда посылает датаграмму, которую примут все станцнн, выдавшие команду NB_ReceiveBroadcastDatagram (в том чнсле и передающая станция, если она тоже выдала команду NB_ReceiveBroadcastDatagram).

Если на одной станции команда NB_ReceiveBroadcastDatagram выдана несколько раз, все буферы после приема данных будут содержать одну н ту же информацию.

Заметим, что датаграммы, посылаемые этой командой одновременно всем станциям, могут быть прнняты только темн станциями, которые выдали команду NB_ReceiveBroadcastDatagram. Поэтому еслн станция желает принимать датаграммы, передаваемые в "широковещательном" режиме, она должна спецнально к этому подготовиться. В протоколе IPX (в отличне от протокола NETBIOS) существует одна универсальная функция, которая может принимать н обычные, н "широковещательные" датаграммы.

Поля NCB на входе	*Содержимое*
Cmd	0x22
NetworkNameNumber	Номер, прнсвоенный прн добавлении нмени
Buffer	Адрес буфера, содержащего передаваемые данные
Size	Размер буфера

PostRoutine	Дальний указатель на POST-программу илн нулевое значение, еслн POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значенне в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состоянне сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера илн другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x01	Неправильная длнна буфера
0x03	Неправильный код команды
0x13	Неправильный номер именн
0x21	Интерфейс занят
0x22	Выдано слншком много команд
0x23	Неправильное значенне в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера илн другого сетевого аппаратного обеспечения

NB_SendBroadcastDatagram (0xA2)

Команда аналогична предыдущей, за нсключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0xA2.

NB_WReceiveDatagram (0x21)

Команда предназначена для прнема датаграмм, переданных командой NB_SendDatagram. Она не может принимать датаграммы, переданные в "широ-

ковещательном" режиме командой NB_SendBroadcastDatagram. Однако эта команда может принимать датаграммы, посланные на групповое имя.

Если перед вызовом команды в поле NetworkNameNumber блока NCB записать значение 0xFF, команда сможет принимать датаграммы от любой станции для любого имени.

Если длина принятой датаграммы превышает значение, указанное в поле Size, принятый блок данных будет обрезан.

Поля NCB на входе	*Содержимое*
Cmd	0x21
NetworkNameNumber	Номер, присвоенный при добавлении имени или 0xFF
Buffer	Адрес буфера, содержащего передаваемые данные
Size	Размер буфера
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CallName	Имя станции, от которой получена датаграмма
Size	Размер принятого блока данных
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x01	Неправильная длина буфера
0x03	Неправильный код команды
0x06	Размер буфера слишком мал для того, чтобы разместить в нем принятые данные

0x0B	Команда отменеиа
0x13	Неправильный номер имеин
0x17	Имя удалено
0x19	Коифликт имеи (внутреиняя ошибка NETBIOS)
0x21	Интерфейс занят
0x22	Выдано слншком много команд
0x23	Неправильиое зиачеине в поле иомера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояиие сетн (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера илн другого сетевого аппаратного обеспечеиия

NB_ReceiveDatagram (0xA1)

Команда аналогична предыдущей, за нсключеинем того, что она выполняется без ожидания н в поле Cmd необходимо записать зиачение 0xA1.

NB_WReceiveBroadcastDatagram (0x23)

Команда предназначена для прнема датаграмм, переданных командой NB_SendBroadcastDatagram. Оиа не может прниимать датаграммы, переданные командой NB_SendDatagram.

Еслн перед вызовом команды в поле NetworkNameNumber блока NCB записать зиачеине 0xFF, команда сможет прииимать датаграммы от любой станции для любого имеии.

Если длниа принятой датаграммы превышает значение, указаниое в поле Size, прниятый блок данных будет обрезан.

Поля NCB на входе	*Содержимое*
Cmd	0x23
NetworkNameNumber	Номер, присвоенный прн добавленнн нменн или 0xFF
Buffer	Адрес буфера, содержащего передаваемые данные
Size	Размер буфера
PostRoutine	Дальний указатель на POST-программу или нулевое зиачеиие, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CallName	Имя станцни, от которой получена датаграмма
Size	Размер прннятого блока данных
CCode	Промежуточиый код завершения команды
FinalCCode	Окоичательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x01	Неправильная длина буфера
0x03	Неправильный код команды
0x06	Размер буфера слишком мал для того, чтобы разместить в нем приятые данные
0x0B	Команда отменена
0x13	Неправильный номер имени
0x17	Имя удалено
0x19	Конфликт имен (внутренняя ошибка NETBIOS)
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

NB_ReceiveBroadcastDatagram (0xA3)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0xA3.

4.3.3. Работа с каналами

В группе команд, предназначенных для работы с каналами, есть команды для создания канала, команды для удаления канала, а также команда для определения состояния канала.

Канал создается одновременно двумя станциями, одна из которых при этом находится в режиме приема запроса на создание канала, а другая передает такой запрос.

Вы можете создать канал между любымн двумя именами в сети. Прн работе с каналамн имена нспользуются только для создания каналов. Впоследствин, когда каналы уже будут созданы, для передачн данных нспользуются иомера каиалов, а не имеиа или номера нмен.

Можно создать канал с самим собой, еслн при созданнн канала указать в качестве имеин партнера свое имя.

NB_WCall (0x10)

Команда устанавливает канал между двумя имеиами, заданными в блоке NCB. Эти имена могут относиться к программам, работающим на разных станциях илн иа одиой станции. В поле OurName указывается имя станции, которая устаиавливает канал, в поле CallName - имя станции, с которой устанавливается канал.

Для успешного создания канала прнннмающая сторона должна выдать команду NB_Listen, которая будет опнсана ниже.

Можно устаиовнть канал ие только с обычиым, ио н с групповым именем. Для этого придется выдать команду NB_WCall иесколько раз, так как за однн вызов создается только однн канал.

Команда NB_WCall делает несколько попыток создать канал н в случае неудачн возвращает код ошнбки.

Прн созданнн канала указывается время тайм-аута для операций прнема и передачи даниых. Еслн команды приема или передачи даниых через каналы ие будут выполнены в течеине времени тайм-аута, оии (команды) будут прерваны. Прн этом счнтается, что канал неработоспособен.

После создания каиала поле LocalSessionNumber будет содержать присвоеиный номер канала. Сохраните его для нспользования в процессе прнема и передачи даниых по каналу.

Поля NCB на входе	*Содержимое*
Cmd	0x10
CallName	Имя, с которым устанавливается канал
OurName	Имя станцин, создающей канал
ReceiveTimeout	Время ожидания приема, в 1/2 с
SendTimeout	Время ожидания передачи, в 1/2 с
PostRoutine	Дальннй указатель на POST-программу илн нулевое значенне, если POST-программа не нспользуется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
LocalSessionNumber	Прнсвоеиный иомер канала
CCode	Промежуточный код завершения команды
FinalCCode	Окоичательиый код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x09	Нет доступных ресурсов
0x15	Неправильное имя
0x21	Интерфейс занят
0x22	Выдано слншком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x05	Истекло время ожидания
0x09	Нет доступных ресурсов
0x0B	Команда отменена
0x11	Переполнилась таблица каналов
0x12	Создание канала отвергнуто
0x14	Нет ответа от станцнн с указанным именем или в сетн нет такого именн
0x15	Неправильное имя
0x18	Ненормальное закрытие канала
0x19	Конфликт имен (внутренняя ошибка NETBIOS)
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значенне в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состоянне сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

NB_Call (0x90)

Команда аналогична предыдущей, за нсключеннем того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0x90.

NB_WListen (0x11)

Команда работает в паре с предыдущей командой и предназначена для организации канала с вызываемой стороны.

В поле CallName блока NCB необходимо указать имя, с которым устанавливается канал. Если в первый байт имени записать символ "*", канал будет установлен с любой вызывающей станцией.

Программа может выдать несколько команд NB_Listen для создания одновременно нескольких каналов.

В случае успешного завершения команда запишет в поле LocalSessionNumber номер созданного канала.

При создании канала необходимо указать время тайм-аута для операций приема и передачи данных через канал. Сама команда NB_WListen не использует тайм-аут. Программа, выдавшая эту команду, будет находиться в состоянии ожидания до тех пор, пока какая-либо станция не пожелает создать с ней канал. Для исключения состояния "зависания" программы лучше использовать вариант NB_Listen этой команды (без ожидания).

Поля NCB на входе	*Содержимое*
Cmd	0x11
CallName	Имя, с которым устанавливается канал. Если в первый байт имени записать символ "*", канал будет установлен с любой вызывающей станцией
OurName	Имя станции, создающей канал с принимающей стороны
ReceiveTimeout	Время ожидания приема, в 1/2 с
SendTimeout	Время ожидания передачи, в 1/2 с
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
LocalSessionNumber	Присвоенный номер канала
CallName	В это поле записывается имя вызвавшей станции, если при установлении канала было указано имя "*"
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x09	Нет доступных ресурсов

0x15	Неправильное нмя
0x21	Интерфейс занят
0x22	Выдано слншком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера илн другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x09	Нет доступных ресурсов
0x0B	Команда отменена
0x11	Переполнилась таблнца каналов
0x15	Неправильное имя
0x17	Указанное нмя было удалено
0x18	Ненормальное закрытне канала
0x19	Конфликт нмен (внутренняя ошибка NETBIOS)
0x21	Интерфейс занят
0x22	Выдано слншком много команд
0x23	Неправильное значенне в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера илн другого сетевого аппаратного обеспечения

NB_Listen (0x91)

Команда аналогична предыдущей, за исключеннем того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0x91.

NB_WHangUp (0x12)

Команда предназначена для закрытия канала, номер которого указан в поле LocalSessionNumber блока NCB. Она должна быть выполнена с обеих сторон канала после завершения работы.

Еслн для закрываемого с помощью этой команды канала на других станциях выдана команда NB_WReceive, она завершается с кодом ощибкн 0x0A. Выданная для закрываемого канала команда NB_WSend также завершается с кодом 0x0A, но через 20 секунд, которые отводятся ей для завершения своей работы. Еслн команда NB_WSend не успела завершить передачу за 20 секунд, команда NB_WHangUp завершается с кодом 0x05.

Аналогично с кодом 0x0A завершает свою работу и команда NB_WReceiveAny, выданная для закрываемого канала.

Другие команды, выданные для закрываемого канала, завершаются с кодом 0x18.

Если программа пытается закрыть канал, который был уже закрыт или не существовал, такая ситуация не считается ошибочной и в поле кода ошибки проставляется нулевое значение.

Поля NCB на входе	*Содержимое*
Cmd	0x12
LocalSessionNumber	Номер закрываемого канала
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x05	Истекло время ожидания
0x08	Неправильный номер канала
0x0A	Канал уже закрыт
0x0B	Команда отменена
0x18	Ненормальное закрытие канала
0x19	Конфликт имен (внутренняя ошибка NETBIOS)
0x21	Интерфейс занят
0x22	Выдано слишком много команд

0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

NB_HangUp (0x92)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0x92.

NB_WSessionStatus (0x34)

Команда возвращает программе состояние канала, имя которого указано в поле OurName блока NCB. В качестве имени можно указать символ "*", в этом случае программа получит информацию о каналах, относящихся ко всем именам станций, имеющихся в локальной таблице имен на вызывающей команду станции.

Информация о состоянии каналов возвращается в буфер, адрес которого программа должна записать в поле Buffer блока NCB. Размер буфера должен быть указан в поле Size блока NCB.

Формат буфера можно описать следующей структурой:

```
struct _SESSION_STATUS {
     unsigned char NameNumber;
     unsigned char SessionCount;
     unsigned char DatagramsOutstanding;
     unsigned char ReceiveAnyoutstanding;
     struct _SESSION {
          unsigned char LocalSessionNumber;
          unsigned char State;
          char LocalName[16];
          char RemoteName[16];
          unsigned char ReceiveCount;
          unsigned char SendCount;
     } Session[40];
};
```

Приведем список полей буфера:

Название поля	*Назначение поля*
NameNumber	Номер имени канала
SessionCount	Количество каналов
DatagramsOutstanding	Количество выданных команд на прием датаграмм
ReceiveAnyoutstanding	Количество выданных команд на прием командой NB_ReceiveAny
Session	Массив структур, описывающих каждый канал в отдельности
LocalSessionNumber	Номер канала

State	Состояние канала: 1 - ожидание завершения команды NB_Listen; 2 - ожидание завершения команды NB_Call; 3 - канал установлен; 4 - ожидание завершения команды NB_HangUp; 5 - команда NB_HangUp завершила свое выполнение; 6 - канал закрыт с ошибкой.
LocalName	Имя локальной станции
RemoteName	Имя удаленной станции
ReceiveCount	Количество ожидающих завершение команд NB_Receive
SendCount	Количество ожидающих завершения команд NB_Send

Поля NCB на входе	*Содержимое*
Cmd	0x34
Buffer	Дальний адрес буфера для прнема нформацнн о состоянии каналов
Size	Размер буфера
OurNames	Имя канала, для которого необходимо получить нформацню о состоянии. В качестве имени можно указывать "*"
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
Size	Размер заполненной части буфера
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x15	Неправильное нмя
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x01	Неправильиая длииа буфера
0x03	Неправильиый код команды
0x06	Слишком мал размер выделеииого буфера
0x19	Коифликт имеи (внутреиняя ошибка NETBIOS)
0x21	Интерфейс занят
0x22	Выдано слишком миого команд
0x23	Неправильиое зиачеиие в поле иомера адаптера AdapterNumber
0x40 - 0x4F	Необычиое состояиие сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечеиия

NB_SessionStatus (0xB4)

Команда аналогичиа предыдущей, за исключеиием того, что оиа выполняется без ожидания и в поле Cmd иеобходимо записать зиачеиие 0xB4.

4.3.4. Прием и передача данных через каналы

NB_WSend (0x14)

С помощью этой команды программа может передать блок даиых размером от 1 до 65535 байт по созданиому ранее каналу. Перед вызовом команды программа должиа записать иомер канала, по которому будет выполняться передача, в поле LocalSessionNumber блока NCB. Адрес передаваемого блока даиых и его длина должны быть записаны в поля Buffer и Size.

Для приема даиых, передаваемых командой NB_WSend, иеобходимо использовать команду NB_WReceive (или NB_Receive).

Механизм передачи данных с использованием каналов гарантирует ие только доставку блоков даиых, ио и правильную последовательиость, в которой эти блоки будут приняты.

Если истекло время тайм-аута, заданиого при создании канала, команда завершается с ошибкой.

Поля NCB на входе	*Содержимое*
Cmd	0x14
LocalSessionNumber	Номер используемого канала
Buffer	Указатель на буфер, содержащий передаваемые даиые
Size	Размер буфера
PostRoutine	Дальний указатель иа POST-программу или нулевое зиачеиие, если POST-программа ие используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошнбок
0x03	Неправильный код команды
0x21	Интерфейс занят
0x22	Выдано слншком много команд
0x23	Неправильное значенне в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состоянне сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера илн другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x05	Истекло время ожидания
0x08	Неправильный номер канала
0x0A	Канал закрыт
0x0B	Команда отменена
0x18	Ненормальное закрытне канала
0x21	Интерфейс заият
0x22	Выдано слншком много команд
0x23	Неправильное значенне в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состоянне сетн (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера илн другого сетевого аппаратного обеспечения

NB_Send (0x94)

Команда аналогична предыдущей, за нсключеннем того, что она выполняется без ожидания н в поле Cmd необходимо запнсать значение 0x94.

NB_WSendNoAck (0x71)

По своему назначенню команда полностью аналогична предыдущей, однако в отличне от нее не выполняет проверку доставкн блока данных принимающей стороне. За счет этого она работает немного быстрее.

Поля NCB на входе	*Содержимое*
Cmd	0x71
LocalSessionNumber	Номер используемого канала
Buffer	Указатель на буфер, содержащий передаваемые данные
Size	Размер буфера
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x05	Истекло время ожидания
0x08	Неправильный номер канала
0x0A	Канал закрыт
0x0B	Команда отменена
0x18	Ненормальное закрытие канала
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

NB_SendNoAck (0xF1)

Команда аналогичиа предыдущей, за исключеиием того, что оиа выполняется без ожидания и в поле Cmd необходимо записать зиачение 0xF1.

NB_WChainSend (0x17)

Команда работает аналогично команде NB_WSend, однако с ее помощью можно передать сразу два блока данных. Данные передаются как один блок. Общий размер передаваемых с помощью этой команды даниых может достигать 131070 байт.

Первый буфер задается, как и для команды NB_WSend, через поля Buffer и Size. Размер второго буфера должеи быть записан в первые два байта поля CallName блока ECB, а его адрес занимает следующие четыре байта этого поля.

Поля NCB на входе	*Содержимое*
Cmd	0x17
LocalSessionNumber	Номер используемого канала
Buffer	Указатель на первый буфер, содержащий передаваемые данные
Size	Размер первого буфера
CallName	Первые два байта содержат размер второго буфера, следующие четыре байта - дальний адрес второго буфера
PostRoutine	Дальний указатель на POST-программу или нулевое зиачеиие, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточиый код завершеиия команды
FinalCCode	Окончательный код завершеиия команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильиый код команды
0x21	Иитерфейс занят
0x22	Выдано слишком миого команд
0x23	Неправильиое зиачение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычиое состояиие сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x05	Истекло время ожидания
0x08	Неправильный иомер канала
0x0A	Канал закрыт
0x0B	Команда отмеиена
0x18	Неиормальиое закрытие канала
0x21	Интерфейс занят
0x22	Выдано слишком миого команд
0x23	Неправильиое зиачение в поле иомера адаптера AdapterNumber
0x40 - 0x4F	Необычиое состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера илн другого сетевого аппаратного обеспечеиия

NB_ChainSend (0x97)

Команда аналогичиа предыдущей, за исключеинем того, что оиа выполняется без ожидания и в поле Cmd необходимо записать зиачение 0x97.

NB_WChainSendNoAck (0x72)

Команда аналогичиа команде NB_WCainSend, однако в отличие от иее ие выполияет проверку доставки блока даниых прииимающей стороие. За счет этого она работает иемиого быстрее.

Поля NCB на входе	*Содержимое*
Cmd	0x72
LocalSessionNumber	Номер используемого канала
Buffer	Указатель на первый буфер, содержащий передаваемые данные
Size	Размер первого буфера
CallName	Первые два байта содержат размер второго буфера, следующие четыре байта - дальиий адрес второго буфера
PostRoutine	Дальний указатель иа POST-программу или нулевое зиачение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточиый код завершеиия команды
FinalCCode	Окончательиый код завершеиия команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильиый код команды
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильиое зиачение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояине сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x05	Истекло время ожидания
0x08	Неправильиый иомер канала
0x0A	Канал закрыт
0x0B	Команда отмеиена
0x18	Неиормальиое закрытие канала
0x21	Интерфейс занят
0x22	Выдано слишком миого команд
0x23	Неправильное зиачеине в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычиое состояиие сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратиого обеспечеиия

NB_ChainSendNoAck (0xF2)

Команда аналогична предыдущей, за исключением того, что оиа выполняется без ожидания н в поле Cmd иеобходимо запнсать значение 0xF2.

NB_WReceive (0x15)

Команда принимает даниые, посланные командами NB_WSend или NB_WChainSend.

Еслн размер буфера иедостаточеи для запнси принятых данных, команда возвращает код ошибкн 0x06; в этом случае вы можете вызвать команду еще раз для того, чтобы прочесть даниые, ие поместившиеся в буфере при предыдущем вызове команды.

Поля NCB на входе	*Содержимое*
Cmd	0x15
LocalSessionNumber	Номер используемого канала
Buffer	Указатель на буфер, используемый для записи принятых данных
Size	Размер буфера
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
Size	Количество принятых и записанных в буфер байт данных
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x21	Интерфейс занят
0x22	Выдано слишком много команд
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x05	Истекло время ожидания
0x06	Размер буфера недостаточен для записи принятых данных
0x08	Неправильный номер канала
0x0A	Канал закрыт
0x0B	Команда отменена
0x18	Ненормальное закрытие канала
0x21	Интерфейс занят
0x22	Выдано слишком много команд

0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

NB_Receive (0x95)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0x95.

NB_WReceiveAny (0x16)

Команда принимает данные от всех партнеров. Для этой команды вместо номера канала необходимо указать номер имени, полученный вашей программой при добавлении имени. Если в поле NetworkNameNumber проставить значение 0xFF, эта команда будет принимать данные от любых партнеров для любых каналов, созданных на вашей станции.

Если размер буфера недостаточен для записи принятых данных, команда возвращает код ошибки 0x06; в этом случае вы можете вызвать команду еще раз для того, чтобы прочесть данные, не поместившиеся в буфере при предыдущем вызове команды.

Поля NCB на входе	*Содержимое*
Cmd	0x16
NetworkNameNumber	Номер имени или 0xFF
Buffer	Указатель на буфер, используемый для записи принятых данных
Size	Размер буфера
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
Size	Количество принятых и записанных в буфер байт данных
NetworkNameNumber	Номер имени станции, от которой пришли данные
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x21	Интерфейс занят
0x22	Выдано слишком много команд

0x23	Неправильное значенне в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состоянне сетн (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошнбок
0x03	Неправильный код команды
0x06	Размер буфера недостаточен для запнсн прннятых данных
0x08	Неправильный номер канала
0x0A	Канал закрыт
0x0B	Команда отменена
0x13	Неправильный номер нмени
0x17	Указанное нмя было удалено
0x18	Ненормальное закрытие канала
0x19	Конфликт имен (внутренняя ошнбка NETBIOS)
0x21	Интерфейс занят
0x22	Выдано слншком много команд
0x23	Неправильное значенне в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состоянне сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера илн другого сетевого аппаратного обеспечения

NB_ReceiveAny (0x96)

Команда аналогична предыдущей, за нсключеннем того, что она выполняется без ожидания и в поле Cmd необходимо запнсать значенне 0x96.

4.3.5. Другие команды

В этом разделе мы опишем команды NETBIOS, позволяющне сбросить драйвер NETBIOS в нсходное состоянне, отменить выданную ранее команду.

NB_WResetAdapter (0x32)

Команда сбрасывает NETBIOS в исходное состояние, удаляет все имеющиеся каналы и имена (кроме постоянного именн, которое нельзя удалить, не вытащив сетевой адаптер из компьютера). С помощью этой команды можно также изменить максимальное количество доступных программе каналов и нспользуемых одновременно блоков NCB. По умолчанню доступны шесть каналов н 12 блоков NCB.

Поля NCB на входе	*Содержимое*
Cmd	0x32
LocalSessionNumber	Максимальное количество каналов или 0 для использования значения по умолчанию
NetworkNameNumber	Максимальное количество блоков NCB или 0 для использования значения по умолчанию
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	Не используется

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

NB_WCancel (0x35)

Команда используется для отмены других запущенных команд. Адрес NCB для отменяемой команды должен быть записан в поле Buffer.

С помощью этой команды нельзя отменить следующие команды: NB_AddName, NB_AddGroupName, NB_DeleteName, NB_SendDatagramm, NB_SendBroadcastDatagramm, NB_ResetAdapter, NB_SessionStatus, NB_Cancel, NB_Unlink.

Поля NCB на входе	*Содержимое*
Cmd	0x35
Buffer	Указатель на блок NCB, для которого отменяется команда
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
FinalCCode	Окончательный код завершения команды

Содержимое поля CCode на выходе	Не используется

Содержимое поля FinalCCode на выходе	*Значение*
0x00	Нет ошибок
0x03	Неправильный код команды
0x23	Неправильное значение в поле номера адаптера AdapterNumber
0x24	Команда уже начала выполняться, когда пришел запрос на ее отмену
0x26	Данную команду нельзя отменить
0x40 - 0x4F	Необычное состояние сети (Unusual network condition)
0x50 - 0xFE	Сбой сетевого адаптера или другого сетевого аппаратного обеспечения

4.4. Коды ошибок

Приведем список кодов ошибок, возвращаемых NETBIOS:

Код ошибки	*Значение*
00	Команда выполнена без ошибок
01	Неправильная длина буфера. Возможно, что при выполнении операции передачи датаграммы указан размер буфера более 512 байт
03	Неправильная команда
05	Истекло заданное время ожидания. Истекло время ожидания для команды NB_Call. Другая возможная причина - истекло время ожидания, заданное при создании канала для команд приема или передачи данных по каналу
06	Данные приняты не полностью. Указанный размер буфера недостаточен для размещения принятых данных
07	Данные, переданные в режиме без подтверждения, не были приняты
08	Неправильный номер канала. Канал с указанным номером не существует
09	Нет доступных ресурсов
0A	Канал закрыт
0B	Команда отменена
0D	Такое имя уже есть в локальной таблице имен
0E	Переполнение локальной таблицы имен. Задано более 16 имен
0F	Указанное имя используется активным каналом

11	Переполненне локальной таблнцы каналов
12	Созданне канала отвергнуто. Это означает, что на удаленном компьютере не выполняется команда NB_Listen, предназначенная для создания канала
13	Неправильный номер адаптера
14	NETBIOS не может найтн вызываемое имя
15	Имя не найдено илн задано неправильно
16	Имя уже нспользуется удаленной станцней
17	Имя удалено
18	Ненормальное завершенне работы канала. Лнбо удаленный компьютер был выключен, лнбо возниклн не-нсправностн в его сетевом оборудованнн, лнбо былн прерваны илн отменены выполнявшнеся там команды передачи данных по каналу
19	Конфликт именн. Драйвер NETBIOS обнаружил прнсутствне в сетн двух однпаковых имен, что недопустнмо
1A	Принят нестандартный пакет, не соответствующий протоколу NETBIOS
21	Интерфейс занят. Эта ошибка появляется прн попытке вызвать команду NETBIOS нз POST-программы илн обработчика прерыванн
22	Слншком много выдано команд
23	Неправильный номер сетевого адаптера. Для номера адаптера можно использовать значения 00h (первый адаптер) илн 01h (второй адаптер)
24	Команда уже начала выполняться, когда прншел запрос на ее отмену
26	Команда не может быть отменена
30	Имя нспользуется другим окруженнем
34	Среда не определена, необходнмо выдать команду NB_Reset
35	Ресурс занят, необходнмо отложнть запрос
36	Превышено макснмальное число работающих приложенн
38	Запрошенный ресурс недоступен
39	Неправильный адрес блока NCB
3A	Команда NB_Reset не может выдаваться нз POST-программы
3C	NETBIOS не смог заблокировать память программы пользователя
3F	Ошибка прн открытин драйвера сетевого адаптера
4E - 4F	Ошнбка в состоянни сетн
F6 - FA	Ошнбка адаптера
FB	Драйвер NETBIOS не загружен
FC	Ошнбка прн открытин сетевого адаптера
FD	Неожиданное закрытне сетевого адаптера
FE	NETBIOS не актнвен

4.5. Система "клиент–сервер" на базе датаграмм

Приведем пример простейшей системы "клиент–сервер", в которую входят две программы, обменивающиеся датаграммами (листинг 18).

Аналогичная система, сделанная на базе протокола IPX, требовала использования процедуры определения сетевых адресов сервера и клиента. Так как сервер может быть запущен на любой станции в сети, программа-клиент сразу после своего запуска не может знать сетевой адрес сервера. Если вы помните, для определения адреса программы-сервера программа-клиент посылала пакет в "широковещательном" режиме сразу всем станциям в сети. Сервер, приняв этот пакет, отвечал клиенту, и таким образом программы узнавали адрес друг друга. Единственное, что должно быть известно программе-клиенту, - это номер сокета, на котором программа-сервер ожидает прихода пакетов.

При использовании протокола NETBIOS ситуация сильно упрощается. Вполне достаточно, если программа-клиент будет знать имя, которое добавляется сервером и используется для приема пакетов от клиентов.

Программа-сервер начинает свою работу с создания объекта класса NETBIOS_DATAGRAM_SERVER. Конструктор в качестве параметра получает имя, добавляемое сервером и используемое для приема пакетов. В нашем случае это имя "NETBIOS Server".

В процессе своей работы конструктор класса NETBIOS_DATAGRAM_SERVER проверяет наличие интерфейса NETBIOS, добавляет имя, полученное им в качестве параметра, и сохраняет полученный номер имени для дальнейшего использования. После завершения работы программы деструктор класса NETBIOS_DATAGRAM_SERVER удалит добавленное имя.

После проверки возможных ошибок программа-сервер вызывает функцию Receive(), которая ожидает прихода пакета от программы-клиента. Когда пакет будет получен, сервер выводит его содержимое как текстовую строку в стандартный поток вывода и завершает свою работу.

```
// ================================================
// Листинг 18. Сервер NETBIOS, вариант с
//             использованием датаграмм
//
// Файл nbserver.cpp
//
// (C) A. Frolov, 1993
// ================================================

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "netbios.hpp"
```

```
// Класс серверов NETBIOS
class NETBIOS_DATAGRAM_SERVER {
     unsigned errno;
     void interrupt ( *int5C)(...);
public:
// Здесь хранится имя сервера и номер этого имени
     char OurName[16];
     unsigned NetworkNameNumber;
// Конструктор, проверяет наличие NETBIOS и добавляет имя
     NETBIOS_DATAGRAM_SERVER(char *Name) {
// Блок NCB, который будет использован при добавлении имени
// NCB AddNameNCB;
// Проверяем длину имени
          if(strlen(Name) > 15) {
               errno = 0xff;
               return;
          }
          strcpy(OurName, Name);
// Проверяем наличие интерфейса NETBIOS
          int5C = getvect(0x5c);
          errno = 0;
          if(FP_SEG(int5C) == 0x0000 || FP_SEG(int5C) == 0xF000) {
               errno=0xff;
               exit(-1);
          }
// Добавляем имя
          AddNameNCB.WAddName(OurName);
// Запоминаем полученный номер имени
          NetworkNameNumber = AddNameNCB.GetNetworkNameNumber();
          errno = AddNameNCB.Error();
     }
// Деструктор, удаляет имя.
     ~NETBIOS_DATAGRAM_SERVER() {
          NCB AddNameNCB;
          AddNameNCB.WDeleteName(OurName);
          errno = AddNameNCB.Error();
     }
// Функция для проверки кода ошибки
     int Error(void) {return errno;}
// Функция для приема датаграммы
     void Receive(char *ReceiveBuffer, unsigned BufferSize) {
          NCB ReceiveNCB;
```

```
// Записываем в NCB адрес и длину буфера
          ReceiveNCB.SetBuffer(ReceiveBuffer, BufferSize);
// Выполняем прием датаграммы с ожиданием
          ReceiveNCB.WReceiveDatagram(NetworkNameNumber);
     }
};
void main(void) {
// Наш сервер с именем "NETBIOS Server"
     NETBIOS_DATAGRAM_SERVER Server("NETBIOS Server");
     char ReceiveBuffer[512];
// Проверяем, были ли ошибки на этапе инициализации сервера.
     if(Server.Error()) {
          printf("Ошибка %02.2X\n", Server.Error());
          return;
     }
     printf("Инициализация завершена.\n");
     printf("Ожидаем сообщение от клиента.\n");
// Принимаем сообщение от клиента
     Server.Receive(ReceiveBuffer, 512);
     printf("Принято: >%s<\n",ReceiveBuffer);
}
```

Программа-сервер работает в паре с программой-клиентом (листинг 19).

После запуска программа-клиент создает объект класса NETBIOS_DATAGRAM_CLIENT. Конструктор и деструктор этого класса выполняют действия, аналогичные конструктору и деструктору класса NETBIOS_DATAGRAM_SERVER.

После инициализации и проверки ошибок программа-клиент посылает сообщение "Привет от клиента NETBIOS!" серверу с именем "NETBIOS Server" при помощи функции Send(). Затем программа-клиент завершает свою работу.

```
// ================================================
// Листинг 19. Клиент NETBIOS, вариант с
//           использованием датаграмм
//
// Файл nbclient.cpp
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
```

```
#include "netbios.hpp"
// Класс клиентов NETBIOS
class NETBIOS_DATAGRAM_CLIENT {
     unsigned errno;
public:
// Здесь хранится имя клиента и номер этого имени
     char OurName[16];
     unsigned NetworkNameNumber;
     union REGS regs;
// Конструктор, проверяет наличие NETBIOS и добавляет имя
     NETBIOS_DATAGRAM_CLIENT(char *Name) {
// Блок NCB, который будет использован при добавлении имени
          NCB AddNameNCB;
// Проверяем длину имени имя
          if(strlen(Name) > 15) {
               errno = 0xff;
               return;
          }
          strcpy(OurName, Name);
// Проверяем наличие интерфейса NETBIOS
          int5C = getvect(0x5c);
          errno = 0;
          if(FP_SEG(int5C) == 0x0000 || FP_SEG(int5C) == 0xF000) {
               errno=0xff;
               exit(-1);
          }
// Добавляем имя
          AddNameNCB.WAddName(OurName);
// Запоминаем полученный номер имени
          NetworkNameNumber = AddNameNCB.GetNetworkNameNumber();
          errno = AddNameNCB.Error();
     }
// Деструктор, удаляет имя.
     ~NETBIOS_DATAGRAM_CLIENT() {
          NCB AddNameNCB;
          AddNameNCB.WDeleteName(OurName);
          errno = AddNameNCB.Error();
     }
// Функция для проверки кода ошибки
     int Error(void) {return errno;}
```

```
// Функция для приема датаграммы
     void Receive(char *ReceiveBuffer, unsigned BufferSize) {
          NCB ReceiveNCB;
// Записываем в NCB адрес и длину буфера
          ReceiveNCB.SetBuffer(ReceiveBuffer, BufferSize);
// Выполияем прием датаграммы с ожиданием
          ReceiveNCB.WReceiveDatagram(NetworkNameNumber);
     }
// Функция для передачи датаграммы
     void Send(char *ReceiveBuffer, unsigned BufferSize,
                              char *CallName) {
          NCB SendNCB;
// Устанавливаем адрес и длину буфера
          SendNCB.SetBuffer(ReceiveBuffer, BufferSize);
// Устанавливаем имя партнера, которому будет передана
// наша датаграмма
          SendNCB.SetCallName(CallName);
// Передаем датаграмму с ожиданием
          SendNCB.WSendDatagram(NetworkNameNumber);
     }
};
void main(void) {
// Наш клиент с именем "NETBIOS Client"
     NETBIOS_DATAGRAM_CLIENT Client("NETBIOS Client");
// Проверяем, были ли ошибки на этапе инициализации клиента.
     if(Client.Error()) {
          printf("Ошибка %02.2X\n", Client.Error());
          return;
     }
     printf("Инициализация завершена.\n");
// Передаем сообщение серверу с именем "NETBIOS Server"
     Client.Send("Привет от клиента NETBIOS!", 512, "NETBIOS
                    Server");
}
```

В include-файле netbios.hpp (листинг 20) приведены определения констант и классов для работы с протоколом NETBIOS через прерывание INT 5Ch.

В классе NCB, кроме структуры данных _NCB, определены конструктор NCB() и несколько других функций для работы с этим классом.

Конструктор расписывает структуру ncb нулями и сбрасывает код ошибки в переменной errno.

Функция NetBios() вызывает прерывание NETBIOS.

Функции WAddName() и WDeleteName() определены в файле nbfunc.cpp (листинг 21). Они предназначены для добавления и удаления имени.

Назначение остальных функций вы можете узнать, прочитав комментарии к программе в листинге 20.

```
// ================================================
// Листинг 20. Классы для работы с NETBIOS
//
// Файл netbios.hpp
//
// (C) A. Frolov, 1993
// ================================================
// Команды NETBIOS
// Команды для работы с именами
#define NB_WAddName 0x30
#define NB_AddName 0xb0
#define NB_WAddGroupName 0x36
#define NB_AddGroupName 0xb6
#define NB_WDeleteName 0x31
#define NB_DeleteName 0xb1
// Команды для передачи датаграмм
#define NB_WSendDatagram 0x20
#define NB_SendDatagram 0xa0
#define NB_WSendBroadcastDatagram 0x22
#define NB_SendBroadcastDatagram 0xa2
// Команды для приема датаграмм
#define NB_WReceiveDatagram 0x21
#define NB_ReceiveDatagram 0xa1
#define NB_WReceiveBroadcastDatagram 0x23
#define NB_ReceiveBroadcastDatagram 0xa3
// Команды для работы с каналами
#define NB_WCall 0x10
#define NB_Call 0x90
#define NB_WListen 0x11
#define NB_Listen 0x91
#define NB_WHangUp 0x12
#define NB_HangUp 0x92
// Команды для передачи данных по каналу
#define NB_WSend 0x14
#define NB_Send 0x94
#define NB_WSendNoAck 0x71
#define NB_SendNoAck 0xf1
#define NB_WChainSend 0x17
#define NB_ChainSend 0x97
#define NB_WChainSendNoAck 0x72
#define NB_ChainSendNoAck 0xf2
```

```
// Команды для приема данных по каналу
#define NB_WReceive 0x15
#define NB_Receive 0x95
#define NB_WReceiveAny 0x16
#define NB_ReceiveAny 0x96
// Прочие команды
#define NB_WResetAdapter 0x32
#define NB_WCancel 0x35
#define NB_WSessionStatus 0x34
#define NB_SessionStatus 0xb4
// Класс NCB для работы с командами NETBIOS
class NCB {
// Стандартный блок NCB в формате NETBIOS
     struct _NCB {
          unsigned char Cmd;
          unsigned char CCode;
          unsigned char LocalSessionNumber;
          unsigned char NetworkNameNumber;
          void far *Buffer;
          unsigned Size;
          char CallName[16];
          char OurName[16];
          unsigned char ReceiveTimeout;
          unsigned char SendTimeout;
          void interrupt (*PostRoutine)(void);
          unsigned char AdapterNumber;
          unsigned char FinalCCode;
          unsigned char Reserved[14];
     } ncb;
     struct SREGS sregs;
     union REGS regs;
     unsigned errno;
// Функция для вызова NETBIOS
     void NetBios(void) {
          sregs.es = FP_SEG(&ncb);
          regs.x.bx = FP_OFF(&ncb);
          int86x(0x5c, &regs, &regs, &sregs);
     }
public:
// Конструктор, расписывает ncb нулями
     NCB() {
          memset(&ncb, 0, sizeof(ncb));
          errno = 0;
     }
```

```
// Функция возвращает код ошибки
     int Error(void) {return errno;}
// Функция для добавления имени
     void WAddName(char *name);
// Функция для удаления имени
     void WDeleteName(char *name);
// Функция для определения номера имени
     unsigned GetNetworkNameNumber(void)
          { return(ncb.NetworkNameNumber); }
// Функция для установки адреса и размера буфера
     void SetBuffer(char far *Buf,   unsigned BufSize) {
          ncb.Buffer = Buf;
          ncb.Size = BufSize;
     }
// Установка в ncb имени вызываемого партнера
     void SetCallName(char *name);
// Прием датаграмм с ожиданием
     void WReceiveDatagram(int NetwrkNameNumber) {
// Заполняем поле номера своего имени
          ncb.NetworkNameNumber = NetwrkNameNumber;
// Вызываем NETBIOS
          ncb.Cmd = NB_WReceiveDatagram;
          NetBios();
     }
// Передача датаграмм с ожиданием
     void WSendDatagram(int NetwrkNameNumber) {
// Заполняем поле номера своего имени
          ncb.NetworkNameNumber = NetwrkNameNumber;
          ncb.Cmd = NB_WSendDatagram;
// Вызываем NETBIOS
          NetBios();
     }
};
```

В файле nbfunc.cpp приведены определения некоторых функций из класса NCB (листинг 21):

```
// ================================================
// Листинг 21. Функции для NETBIOS
//
// Файл nbfunc.cpp
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "netbios.hpp"
// Добавляем имя
     void NCB::WAddName(char *name) {
          char buf[16];
// Проверяем длину имени
          if(strlen(name) > 15) {
               errno = 0xff;
               return;
          }
          strcpy(buf, name);
// При необходимости дополняем имя пробелами
          while (strlen(buf) < 15) strcat(buf, " ");
// Вызываем NETBIOS
          ncb.Cmd = NB_WAddName;
          strcpy(ncb.OurName, buf);
          NetBios();
          errno = ncb.FinalCCode;
     }
// Удаление имени
     void NCB::WDeleteName(char *name) {
          char buf[16];
// Проверяем длину имени
          if(strlen(name) > 15) {
               errno = 0xff;
               return;
          }
          strcpy(buf, name);
// При необходимости дополняем имя пробелами
```

```
            while (strlen(buf) < 15) strcat(buf, " ");
            strcpy(ncb.OurName, buf);
// Вызываем NETBIOS
            ncb.Cmd = NB_WDeleteName;
            NetBios();
            errno = ncb.FinalCCode;
      }
// Установка имени вызываемого партнера
      void NCB::SetCallName(char *name) {
            char buf[16];
            if(strlen(name) > 15) {
                  errno = 0xff;
                  return;
            }
            strcpy(buf, name);
            while (strlen(buf) < 15) strcat(buf, " ");
            strcpy(ncb.CallName, buf);
      }
```

4.6. Система "клиент–сервер" на базе каналов

Приведем пример системы "клиент–сервер", реализованной с использованием каналов протокола NETBIOS (листинг 22).

После запуска программа-сервер создает объект класса NETBIOS_SESSION_SERVER. Конструктор этого объекта проверяет присутствие интерфейса NETBIOS, добавляет имя, переданное ему в качестве параметра, затем создает канал при помощи функции WListen().

Деструктор класса NETBIOS_SESSION_SERVER перед удалением имени удаляет канал, так как имя нельзя удалить, если оно используется каким-либо каналом.

После того, как отработал конструктор, программа-сервер проверяет ошибки и вызывает функцию Receive(), которая ожидает приема данных по созданному каналу. После приема сервер отображает принятые данные как текстовую строку и завершает свою работу.

```
// ================================================
// Листинг 22. Сервер NETBIOS, вариант с
//             использованием каналов
//
// Файл nbserver.cpp
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
```

```
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "netbios.hpp"
// Класс серверов NETBIOS
class NETBIOS_SESSION_SERVER {
     unsigned errno;
     void interrupt ( *int5C)(...);
public:
// Здесь хранится имя сервера и номер этого имени
     char OurName[16];
     unsigned NetworkNameNumber;
// Блок NCB, который будет использован при добавлении имени
     NCB AddNameNCB;
// Конструктор, проверяет наличие NETBIOS и добавляет имя
     NETBIOS_SESSION_SERVER(char *Name) {
// Проверяем длину имени имя
          if(strlen(Name) > 15) {
               errno = 0xff;
               return;
          }
          strcpy(OurName, Name);
// Проверяем наличие интерфейса NETBIOS
          int5C = getvect(0x5c);
          errno = 0;
          if(FP_SEG(int5C) == 0x0000 || FP_SEG(int5C) == 0xF000) {
               errno=0xff;
               exit(-1);
          }
// Добавляем имя
          AddNameNCB.WAddName(OurName);
// Запоминаем полученный номер имени
          NetworkNameNumber = AddNameNCB.GetNetworkNameNumber();
          errno = AddNameNCB.Error();
          if(errno) return;
// Устанавливаем "*" в поле CallName, это означает,
// что сервер будет обрабатывать запросы на создание
// канала от любого имени
          AddNameNCB.SetCallName("*");
// Устанавливаем время тайм-аута для команд
// приема и передачи данных по каналу
          AddNameNCB.SetRtoSto(20,20);
```

```
// Создаем канал с принимающей стороны
            AddNameNCB.WListen();
      }
// Деструктор, удаляет канал и имя.
      ~NETBIOS_SESSION_SERVER() {
// Удаление канала
            AddNameNCB.WHangUp();
// Удаление имени
            AddNameNCB.WDeleteName(OurName);
            errno = AddNameNCB.Error();
      }
// Функция для проверки кода ошибки
      int Error(void) {return errno;}
// Функция для приема данных по каналу
      void Receive(char *ReceiveBuffer, unsigned BufferSize) {
// Записываем в NCB адрес и длину буфера
            AddNameNCB.SetBuffer(ReceiveBuffer, BufferSize);
// Выполняем прием датаграммы с ожиданием
            AddNameNCB.WReceive();
      }
};
void main(void) {
// Наш сервер с именем "NETBIOS Server"
      NETBIOS_SESSION_SERVER Server("NETBIOS Server");
      char ReceiveBuffer[512];
// Проверяем, были ли ошибки на этапе инициализации сервера.
      if(Server.Error()) {
            printf("Ошибка %02.2X\n", Server.Error());
            return;
      }
      printf("Инициализация завершена.\n");
      printf("Ожидаем сообщение от клиента.\n");
// Принимаем сообщение от клиента по каналу, который был создан
// конструктором класса NETBIOS_SESSION_SERVER
      Server.Receive(ReceiveBuffer, 512);
      printf("Принято: >%s<\n",ReceiveBuffer);
}
```

В файле nbclient.cpp иаходится программа-клиент (листинг 23), работающая в паре с только что прнведенной программой-сервером.

Программа-клиент создает объект NETBIOS_SESSION_CLIENT, конструктор которого выполняет действия, аналогичные коиструктору класса

NETBIOS_SESSION_SERVER. Есть одно отличие: для создания канала в конструкторе класса NETBIOS_SESSION_CLIENT используется функция WCall(), а не WListen(). Конструктор создает канал с программой-сервером, указывая имя "NETBIOS Server", которое используется сервером для работы с клиентом.

После проверки ошибок программа-клиент с помощью функции Send() передает по созданному каналу программе-серверу сообщение "Привет от клнента NETBIOS!" и завершает свою работу.

```
// ================================================
// Листинг 23. Клиент NETBIOS, вариант с
//             использованием каналов
//
// Файл nbclient.cpp
//
// (C) A. Frolov, 1993
// ================================================
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include "netbios.hpp"
// Класс клиентов NETBIOS
class NETBIOS_SESSION_CLIENT {
     unsigned errno;
     void interrupt ( *int5C)(...);
// Блок NCB, который будет использован при добавлении имени
          NCB AddNameNCB;
public:
// Здесь хранится имя клиента и номер этого имени
     char OurName[16];
     unsigned NetworkNameNumber;
// Конструктор, проверяет наличие NETBIOS и добавляет имя
     NETBIOS_SESSION_CLIENT(char *Name) {

// Проверяем длину имени имя
          if(strlen(Name) > 15) {
               errno = 0xff;
               return;
          }
          strcpy(OurName, Name);
// Проверяем наличие интерфейса NETBIOS
          int5C = getvect(0x5c);
```

```
            errno = 0;
            if(FP_SEG(int5C) == 0x0000 || FP_SEG(int5C) == 0xF000) {
                  errno=0xff;
                  exit(-1);
            }
// Добавляем имя
            AddNameNCB.WAddName(OurName);
// Запоминаем полученный номер имени
            NetworkNameNumber = AddNameNCB.GetNetworkNameNumber();
// Если при добавлении имени были ошибки,
// завершаем работу программы
            errno = AddNameNCB.Error();
            if(errno) return;
// Устанавливаем имя сервера, с которым будем создавать канал
            AddNameNCB.SetCallName("NETBIOS Server");
// Устанавливаем время тайм-аута
// для передачи и приема данных по каналу
            AddNameNCB.SetRtoSto(20,20);
// Устанавливаем канал с передающей стороны
            AddNameNCB.WCall();
      }
// Деструктор, удаляет канал и имя.
      ~NETBIOS_SESSION_CLIENT() {
// Удаление канала
            AddNameNCB.WHangUp();
// Удаление имени
            AddNameNCB.WDeleteName(OurName);
            errno = AddNameNCB.Error();
      }
// Функция для проверки кода ошибки
      int Error(void) {return errno;}
// Функция для передачи по каналу
      void Send(char *ReceiveBuffer, unsigned BufferSize) {
// Устанавливаем адрес и длину буфера
            AddNameNCB.SetBuffer(ReceiveBuffer, BufferSize);
// Передаем данные по каналу с ожиданием
            AddNameNCB.WSend();
      }
};
```

```
void main(void) {
// Ваш клиент с именем "NETBIOS Client"
     NETBIOS_SESSION_CLIENT Client("NETBIOS Client");
// Проверяем, были ли ошибки на этапе инициализации клиента.
     if(Client.Error()) {
          printf("Ошибка %02.2X\n", Client.Error());
          return;
     }
     printf("Инициализация завершена.\n");
// Передаем сообщение серверу по созданному каналу. Канал был
// создан при работе конструктора класса NETBIOS_SESSION_CLIENT.
     Client.Send("Привет от клиента NETBIOS!", 512);
}
```

Файл netbios.hpp (листинг 24) содержит все необходимые определения для программ, работающих с каналами NETBIOS:

```
// ================================================
// Листинг 24. Классы для работы с NETBIOS
//
// Файл netbios.hpp
//
// (C) A. Frolov, 1993
// ================================================
// Команды NETBIOS
// Команды для работы с именами
#define NB_WAddName 0x30
#define NB_AddName 0xb0
#define NB_WAddGroupName 0x36
#define NB_AddGroupName 0xb6
#define NB_WDeleteName 0x31
#define NB_DeleteName 0xb1
// Команды для передачи датаграмм
#define NB_WSendDatagram 0x20
#define NB_SendDatagram 0xa0
#define NB_WSendBroadcastDatagram 0x22
#define NB_SendBroadcastDatagram 0xa2
// Команды для приема датаграмм
#define NB_WReceiveDatagram 0x21
#define NB_ReceiveDatagram 0xa1
#define NB_WReceiveBroadcastDatagram 0x23
#define NB_ReceiveBroadcastDatagram 0xa3
// Команды для работы с каналами
#define NB_WCall 0x10
#define NB_Call 0x90
```

```
#define NB_WListen 0x11
#define NB_Listen 0x91
#define NB_WHangUp 0x12
#define NB_HangUp 0x92
// Команды для передачи данных по каналу
#define NB_WSend 0x14
#define NB_Send 0x94
#define NB_WSendNoAck 0x71
#define NB_SendNoAck 0xf1
#define NB_WChainSend 0x17
#define NB_ChainSend 0x97
#define NB_WChainSendNoAck 0x72
#define NB_ChainSendNoAck 0xf2
// Команды для приема данных по каналу
#define NB_WReceive 0x15
#define NB_Receive 0x95
#define NB_WReceiveAny 0x16
#define NB_ReceiveAny 0x96
// Прочие команды
#define NB_WResetAdapter 0x32
#define NB_WCancel 0x35
#define NB_WSessionStatus 0x34
#define NB_SessionStatus 0xb4
// Класс NCB для работы с командами NETBIOS
class NCB {
// Стандартный блок NCB в формате NETBIOS
     struct _NCB {
          unsigned char Cmd;
          unsigned char CCode;
          unsigned char LocalSessionNumber;
          unsigned char NetworkNameNumber;
          void far *Buffer;
          unsigned Size;
          char CallName[16];
          char OurName[16];
          unsigned char ReceiveTimeout;
          unsigned char SendTimeout;
          void interrupt (*PostRoutine)(void);
          unsigned char AdapterNumber;
          unsigned char FinalCCode;
          unsigned char Reserved[14];
     } ncb;
     struct SREGS sregs;
     union REGS regs;
     unsigned errno;
```

```
// Функция для вызова NETBIOS
     void NetBios(void) {
          sregs.es = FP_SEG(&ncb);
          regs.x.bx = FP_OFF(&ncb);
          int86x(0x5c, &regs, &regs, &sregs);
     }
public:
// Конструктор, расписывает ncb нулями
     NCB() {
          memset(&ncb, 0, sizeof(ncb));
          errno = 0;
     }
// Функция возвращает код ошибки
     int Error(void) {return errno;}
// Функция для добавления имени
     void WAddName(char *name);
// Функция для удаления имени
     void WDeleteName(char *name);
// Функция для определения номера имени
     unsigned GetNetworkNameNumber(void)
          { return(ncb.NetworkNameNumber); }
// Функция для установки адреса и размера буфера
     void SetBuffer(char far *Buf,   unsigned BufSize) {
          ncb.Buffer = Buf;
          ncb.Size = BufSize;
     }
// Установка в NCB тайм-аута
     void SetRtoSto(int rto, int sto) {
          ncb.ReceiveTimeout = rto;
          ncb.SendTimeout = sto;
     }
// Установка в ncb имени вызываемого партнера
     void SetCallName(char *name);
// Установка в ncb имени нашей станции
     void SetOurName(char *name);
// Прием датаграмм с ожиданием
     void WReceiveDatagram(int NetwrkNameNumber) {
// Заполняем поле номера своего имени
          ncb.NetworkNameNumber = NetwrkNameNumber;
// Вызываем NETBIOS
          ncb.Cmd = NB_WReceiveDatagram;
```

```
            NetBios();
      }
// Передача датаграмм с ожиданием
      void WSendDatagram(int NetwrkNameNumber) {
// Заполняем поле номера своего имени
            ncb.NetworkNameNumber = NetwrkNameNumber;
            ncb.Cmd = NB_WSendDatagram;
// Вызываем NETBIOS
            NetBios();
      }
// Создание канала с принимающей стороны
      void WListen(void) {
            ncb.Cmd = NB_WListen;
            NetBios();
      }
// Создание канала с передающей стороны
      void WCall(void) {
            ncb.Cmd = NB_WCall;
            NetBios();
      }
// Удаление канала
      void WHangUp(void) {
            ncb.Cmd = NB_WHangUp;
            NetBios();
      }
// Прием из канала с ожиданием
      void WReceive(void) {
            ncb.Cmd = NB_WReceive;
            NetBios();
      }
// Передача в канал с ожиданием
      void WSend(void) {
            ncb.Cmd = NB_WSend;
            NetBios();
      }
};
```

Приложение 1

ФУНКЦИИ IPX

Функции для работы с сокетами

IPXOpenSocket - открыть сокет

На входе: BX = 00h.
AL = Тип сокета:
00h - короткоживущий;
FFh - долгоживущий.
DX = Запрашиваемый номер сокета или 0000h, если требуется получить динамический номер сокета.
Примечание. Байты номера сокета находятся в перевернутом виде.

На выходе: AL = Код завершения:
00h - сокет открыт;
FFh - этот сокет уже был открыт раньше;
FEh - переполнилась таблица сокетов.
DX = Присвоенный номер сокета.

IPXCloseSocket - закрыть сокет

На входе: BX = 01h.
DX = Номер закрываемого сокета.

На выходе: Регистры не используются.

Функции для работы с сетевыми адресами

IPXGetLocalTaget - получить непосредственный адрес

На входе: BX = 02h.
ES:SI = Указатель на буфер длиной 12 байт, содержащий полный сетевой адрес станции, на которую будет послан пакет.
ES:DI = Указатель на буфер длиной 6 байт, в который будет записан непосредственный адрес, т. е. адрес той станции, которой будет передан пакет. Это может быть адрес моста.

На выходе: AL = Код завершения:
00h - непосредственный адрес был успешно вычислен;
FAh - непосредственный адрес вычислить невозможно, так как к указанной станции нет ни одного пути доступа по сети.
CX = Время пересылки пакета до станции назначения (только если AL равен 0) в тиках системного таймера. Тики таймера следуют с периодом примерно 1/18 секунды.

IPXGetInternetworkAddress - получить собственный адрес

На входе: BX = 09h.
ES:DI = Указатель на буфер длиной 10 байт, в который будет записан адрес станции, на которой работает данная программа. Адрес состоит из номера сети **Network** и адреса станции в сети **Node**.

На выходе: Регистры не используются.

Прием и передача пакетов

IPXListenForPacket - принять IPX-пакет

На входе: BX = 04h.
ES:DI = Указатель на заполненный блок ECB. Необходимо заполнить поля:
ESRAddress;
Socket;
FragmentCnt;
указатели на буферы фрагментов **Address;**
размеры фрагментов **Size.**

На выходе: Регистры не используются.

IPXSendPacket - передать IPX-пакет

На входе: BX = 03h.
ES:DI = Указатель на заполненный блок ECB. Необходимо заполнить поля:
ESRAddress;
Socket;
ImmAddress;
FragmentCnt;
указатели на буферы фрагментов **Address;**
размеры фрагментов **Size.**
В заголовке пакета IPX необходимо заполнить поля:
PacketType;
DestNetwork;
DestNode;
DestSocket.

На выходе: Регистры не используются.

Другие функции IPX и AES

IPXDisconnectFromTaget - отключиться от партиера

На входе: BX = 0Bh.
ES:SI = Указатель иа структуру, содержащую сетевой адрес станции:

```
struct NetworkAddress {
      unsigned char Network[4];
      unsigned char Node[6];
      unsigned char Socket[2];
}
```

На выходе: Регистры ие используются.

IPXSceduleIPXEvent - отложить событие

На входе: BX = 05h.
AX = Время задержки в тиках таймера
ES:SI = Указатель на блок ECB.

На выходе: Регистры не используются.

IPXGetIntervalMarker - получить интервальиый маркер

На входе: BX = 08h.

На выходе: AX = Интервальиый маркер.

IPXCancelEvent - отмеиить событие

На входе: BX = 06h.
ES:SI = Указатель иа блок ECB.

На выходе: AL = Код завершеиия:
00h - функция выполнена без ошибок;
F9h - обработка ECB не может быть отменена;
FFh - указаниый ECB ие используется.

IPXRelinquishControl - выделить время драйверу IPX

На входе: BX = 0Ah.

На выходе: Регистры ие используются.

Приложение 2

ФУНКЦИИ SPX

Инициализация SPX

SPXCheckInstallation - ииицнализировать SPX

На входе: BX = 10h.
AL = 00h.

На выходе: AL = Код завершеиия:
00h - SPX не установлеи;
FFh - SPX установлеи.
BH = Верхнйй (major) иомер версии SPX.
BL = Нижинй (minor) номер версни SPX.
CX = Максимальиое количество каналов SPX, поддерживаемых драйвером SPX.
DX = Количество доступиых каналов SPX.

Образоваиие каиала связи

SPXListenForConnection - создать каиал с прниимающей стороны

На входе: BX = 12h.
AL = Счетчик повторов попыток создать канал связн.
AH = Флаг включения снстемы пернодической проверкн связи (Watchdog Supervision Required Flag).
ES:SI = Указатель иа блок ECB.

На выходе: Регистры не используются.

SPXEstablishConnection - создать каиал с передающей стороиы

На входе: BX = 11h
AL = Счетчик повторов попыток создать канал связн.
AH = Флаг включения системы пернодической проверкн связи (Watchdog Supervision Required Flag).
ES:SI = Указатель иа блок ECB

На выходе: AL = Промежуточный код завершения:
00h - выполняется попытка создать канал;
FFh - указаниый в блоке ECB сокет закрыт;
FDh - сбойиый пакет: лнбо счетчик фрагментов не равеи 1, либо размер фрагмента не равеи 42;
EFh - переполнение локальиой таблицы номеров каналов связи.
DX Присвоенный иомер канала.

Прием и передача пакетов

SPXListenForSequencedPacket - принять SPX-пакет

На входе: BX = 17h.
ES:SI = Указатель на блок ECB.

На выходе: Регистры не используются.

SPXSendSequencedPackett - передать SPX-пакет

На входе: BX = 16h.
ES:SI = Указатель на блок ECB
DX = Номер канала связи.

На выходе: Регистры не используются.

Разрыв канала связи

SPXTerminateConnection - закрыть канал

На входе: BX = 13h
ES:SI = Указатель на блок ECB
DX = Номер канала связи

На выходе: Регистры не используются.

SPXAbortConnection - закрыть канал аварийно

На входе: BX = 14h.
DX = Номер канала связи.

На выходе: Регистры не используются.

Проверка состояния канала

SPXGetConnectionStatus - получить состояние канала

На входе: BX = 15h.
DX = Номер канала связи.
ES:SI = Указатель на буфер размером 44 байта.

На выходе: AL = Код завершения:
00h - канал активен;
EEh - указанный канал не существует.

Приложение 3

ФУНКЦИИ NETBIOS

Работа с именами

NB_WAddName (0x30) - добавить имя

Поля NCB на входе	*Содержимое*
Cmd	0x30
OurName	Добавляемое имя
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
NetworkNameNumber	Присвоенный номер имени
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

NB_AddName (0xB0)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0xB0.

NB_WAddGroupName (0x36) - добавить групповое имя

Поля NCB на входе	*Содержимое*
Cmd	0x36
OurName	Добавляемое групповое имя
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
NetworkNameNumber	Присвоенный номер имени
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

NB_AddGroupName (0xB6)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0xB6.

NB_WDeleteName (0x31) - удалить имя

Поля NCB на входе	*Содержимое*
Cmd	0x31
OurName	Удаляемое имя
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

NB_DeleteName (0xB1)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0xB1.

Прием и передача датаграмм

NB_WSendDatagram (0x20) - послать датаграмму

Поля NCB на входе	*Содержимое*
Cmd	0x20
NetworkNameNumber	Номер, присвоенный при добавлении имени
CallName	Имя станции, которой передаются данные
Buffer	Адрес буфера, содержащего передаваемые данные
Size	Размер буфера
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

NB_SendDatagram (0xA0)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0xA0.

NB_WSendBroadcastDatagram (0x22) - послать датаграмму одновременно всем станциям

Поля NCB на входе	*Содержимое*
Cmd	0x22
NetworkNameNumber	Номер, присвоенный при добавлении имени
Buffer	Адрес буфера, содержащего передаваемые данные
Size	Размер буфера
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

NB_SendBroadcastDatagram (0xA2)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0xA2.

NB_WReceiveDatagram (0x21) - принять датаграмму

Поля NCB на входе	*Содержимое*
Cmd	0x21
NetworkNameNumber	Номер, присвоенный при добавлении имени или 0xFF
Buffer	Адрес буфера, содержащего передаваемые данные
Size	Размер буфера
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CallName	Имя станции, от которой получена датаграмма
Size	Размер принятого блока данных
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

NB_ReceiveDatagram (0xA1)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0xA1.

NB_WReceiveBroadcastDatagram (0x23) - принять датаграмму, посланную одновременно всем станциям в сети

Поля NCB на входе	*Содержимое*
Cmd	0x23
NetworkNameNumber	Номер, прнсвоениый прн добавлеиии имеии или 0xFF
Buffer	Адрес буфера, содержащего передаваемые даниые
Size	Размер буфера
PostRoutine	Дальннй указатель на POST-программу или нулевое зиачеине, еслн POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CallName	Имя станции, от которой получена датаграмма
Size	Размер прннятого блока даниых
CCode	Промежуточный код завершения команды
FinalCCode	Окоичательиый код завершеиия команды

NB_ReceiveBroadcastDatagram (0xA3)

Команда аналогичиа предыдущей, за исключеинем того, что она выполняется без ожидания и в поле Cmd необходимо запнсать зиаченне 0xA3.

Работа с каналами

NB_WCall (0x10) - создать канал с передающей стороиы

Поля NCB на входе	*Содержимое*
Cmd	0x10
CallName	Имя, с которым устанавливается канал
OurName	Имя станцин, создающей канал
ReceiveTimeout	Время ожидания прнема, в 1/2 с
SendTimeout	Время ожидаиия передачи, в 1/2 с
PostRoutine	Дальний указатель на POST-программу или нулевое значенне, еслн POST-програма не нспользуется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
LocalSessionNumber	Присвоеииый иомер канала
CCode	Промежуточиый код завершения комаиды
FinalCCode	Окончательный код завершеиия команды

NB_Call (0x90)

Команда аналогична предыдущей, за исключеннем того, что она выполияется без ожидания н в поле Cmd необходимо записать значение 0x90.

NB_WListen (0x11) - создать канал с прнннмающей стороны

Поля NCB на входе	*Содержимое*
Cmd	0x11
CallName	Имя, с которым устанавливается канал. Еслн в первый байт имени запнсать снмвол "*", канал будет установлен с любой вызывающей станцией
OurName	Имя станции, создающей канал с принимающей стороны
ReceiveTimeout	Время ожидаиия прнема, в 1/2 с
SendTimeout	Время ожидания передачн, в 1/2 с
PostRoutine	Дальиий указатель на POST-программу или нулевое значенне, если POST-программа ие нспользуется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
LocalSessionNumber	Прнсвоенный номер канала
CallName	В это поле записывается имя вызвавшей станцин, если прн установленнн канала было указано имя "*"
CCode	Промежуточиый код завершения команды
FinalCCode	Окончательный код завершеиия команды

NB_Listen (0x91)

Команда аналогична предыдущей, за нсключеннем того, что она выполняется без ожидания н в поле Cmd необходнмо запнсать значенне 0x91.

NB_WHangUp (0x12) - закрыть канал

Поля NCB на входе	*Содержимое*
Cmd	0x12
LocalSessionNumber	Номер закрываемого канала
PostRoutine	Дальннй указатель иа POST-программу или нулевое значение, еслн POST-программа ие используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточиый код завершения команды
FinalCCode	Окончательный код завершения команды

NB_HangUp (0x92)

Команда аналогична предыдущей, за нсключеннем того, что она выполняется без ожидания н в поле Cmd необходимо запнсать значенне 0x92.

NB_WSessionStatus (0x34) - получить состояние канала

Названне поля	*Назначение поля*
NameNumber	Номер именн канала
SessionCount	Количество каналов
DatagramsOutstanding	Количество выданных команд на прнем датаграмм
ReceiveAnyoutstanding	Колнчество выданных команд на прнем командой NB_ReceiveAny
Session	Масснв структур, опнсывающих каждый канал в отдельности
LocalSessionNumber	Номер канала
State	Состоянне канала: 1 - ожиданне завершения команды NB_Listen; 2 - ожиданне завершения команды NB_Call; 3 - канал установлен; 4 - ожиданне завершения команды NB_HangUp; 5 - команда NB_HangUp завершила свое выполненне; 6 - канал закрыт с ошнбкой.
LocalName	Имя локальной станцнн
RemoteName	Имя удаленной станцнн
ReceiveCount	Колнчество ожидающнх завершения комаид NB_Receive
SendCount	Колнчество ожидающнх завершения комаид NB_Send

Поля NCB на входе	*Содержимое*
Cmd	0x34
Buffer	Дальннй адрес буфера для прнема ннформации о состоянин каналов
Size	Размер буфера
OurNames	Имя канала, для которого необходнмо получить ннформацню о состояннн. В качестве именн можно указывать "*"
PostRoutine	Дальннй указатель на POST-программу илн нулевое значенне, еслн POST-программа не нспользуется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
Size	Размер заполиеиной части буфера
CCode	Промежуточный код завершеиия команды
FinalCCode	Окоичательиый код завершеиия команды

NB_SessionStatus (0xB4)

Команда аналогичиа предыдущей, за исключеиием того, что оиа выполняется без ожидания и в поле Cmd иеобходимо записать зиачеиие 0xB4.

Прием и передача данных через каналы

NB_WSend (0x14) - передать даииые через каиал

Поля NCB на входе	*Содержимое*
Cmd	0x14
LocalSessionNumber	Номер используемого канала
Buffer	Указатель иа буфер, содержащий передаваемые даиные
Size	Размер буфера
PostRoutine	Дальиий указатель иа POST-программу или иулевое зиачеиие, если POST-программа ие используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточиый код завершения команды
FinalCCode	Окоичательиый код завершеиия команды

NB_Send (0x94)

Команда аналогичиа предыдущей, за исключеиием того, что она выполняется без ожидания и в поле Cmd иеобходимо записать значеиие 0x94.

NB_WSendNoAck (0x71) - передать даиные через канал без подтверждеиия

Поля NCB на входе	*Содержимое*
Cmd	0x71
LocalSessionNumber	Номер используемого канала
Buffer	Указатель иа буфер, содержащий передаваемые даиные
Size	Размер буфера
PostRoutine	Дальиий указатель на POST-программу или иулевое зиачеиие, если POST-программа ие используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

NB_SendNoAck (0xF1)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0xF1.

NB_WChainSend (0x17) - передать данные через канал, используя сцепленные буферы

Поля NCB на входе	*Содержимое*
Cmd	0x17
LocalSessionNumber	Номер используемого канала
Buffer	Указатель на первый буфер, содержащий передаваемые данные
Size	Размер первого буфера
CallName	Первые два байта содержат размер второго буфера, следующие четыре байта - дальний адрес второго буфера
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

NB_ChainSend (0x97)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0x97.

NB_WChainSendNoAck (0x72) - передать данные через канал, используя сцепленные буферы без подтверждения

Поля NCB на входе	*Содержимое*
Cmd	0x72
LocalSessionNumber	Номер используемого канала
Buffer	Указатель на первый буфер, содержащий передаваемые данные
Size	Размер первого буфера

CallName	Первые два байта содержат размер второго буфера, следующие четыре байта - дальний адрес второго буфера
PostRoutine	Дальний указатель иа POST-программу или нулевое значеиие, если POST-программа ие используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
CCode	Промежуточный код завершеиия команды
FinalCCode	Окончательиый код завершеиия команды

NB_ChainSendNoAck (0xF2)

Команда аналогичиа предыдущей, за исключением того, что оиа выполняется без ожидания и в поле Cmd необходимо записать зиачеиие 0xF2.

NB_WReceive (0x15) - прииять даииые через каиал

Поля NCB на входе	*Содержимое*
Cmd	0x15
LocalSessionNumber	Номер используемого канала
Buffer	Указатель иа буфер, используемый для записи принятых даниых
Size	Размер буфера
PostRoutine	Дальний указатель на POST-программу илн нулевое зиачеиие, если POST-программа ие используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
Size	Количество принятых и записанных в буфер байт данных
CCode	Промежуточиый код завершения команды
FinalCCode	Окончательный код завершенння команды

NB_Receive (0x95)

Команда аналогичиа предыдущей, за исключеиием того, что она выполняется без ожидания и в поле Cmd иеобходимо записать зиачеиие 0x95.

NB_WReceiveAny (0x16) - прииять даииые через канал от любого имеии

Поля NCB на входе	*Содержимое*
Cmd	0x16
NetworkNameNumber	Номер имеии или 0xFF
Buffer	Указатель иа буфер, используемый для записи принятых даниых

Size	Размер буфера
PostRoutine	Дальний указатель на POST-программу или нулевое значение, если POST-программа не используется
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
Size	Количество принятых и записанных в буфер байт данных
NetworkNameNumber	Номер имени станции, от которой пришли данные
CCode	Промежуточный код завершения команды
FinalCCode	Окончательный код завершения команды

NB_ReceiveAny (0x96)

Команда аналогична предыдущей, за исключением того, что она выполняется без ожидания и в поле Cmd необходимо записать значение 0x96.

Другие команды

NB_WResetAdapter (0x32) - сбросить сетевой адаптер

Поля NCB на входе	*Содержимое*
Cmd	0x32
LocalSessionNumber	Максимальное количество каналов или 0 для использования значения по умолчанию
NetworkNameNumber	Максимальное количество блоков NCB или 0 для использования значения по умолчанию
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
FinalCCode	Окончательный код завершения команды

NB_WCancel (0x35) - отменить команду

Поля NCB на входе	*Содержимое*
Cmd	0x35
Buffer	Указатель на блок NCB, для которого отменяется команда
AdapterNumber	Номер адаптера; 0 - первый адаптер, 1 - второй

Поля NCB на выходе	*Содержимое*
FinalCCode	Окончательный код завершения команды

Приложение 4

РАБОТА С IPX, SPX И NETBIOS В СРЕДЕ MS WINDOWS

Созданне программ, спецнально предназначенных для работы в среде Microsoft Windows, не является предметом обсуждения в данной кнпге. Однако мы сделаем несколько замечаннй относнтельио нспользования протоколов IPX, SPX и NETBIOS в программах, работающнх в среде Microsoft Windows версин 3.1 н Microsoft Windows for Workgroups версни 3.1.

Работа в среде Microsoft Windows версни 3.1

Все резидеитные программы, имеющне отношение к сетевой оболочке Novell NetWare, необходимо загружать *до* запуска Windows. Это относится и к эмулятору протокола NETBIOS. Если вы запустнте этн программы из виртуальной машины MS-DOS, работающей в среде WIndows, раио илн поздно произойдет аварнйное завершенне Windows.

Еслн ваша программа работает в среде MS-DOS с протоколамн IPX, SPX илн NETBIOS, оиа без всякнх нзмененнй будет работать н на внртуальиой машине Windows. Вам только иадо проследнть, чтобы все резидеитные программы н эмулятор NETBIOS загружались до запуска Windows.

Еслн же вы желаете создать полиоцениое приложеиие для Windows, работающее с сетевыми протоколами, вам следует нметь в виду, что для всех этих протоколов необходнмо указывать адреса управляющих блоков и буферов, расположенные в первом мегабайте основной оперативной памятн. Кроме того, нз приложений Windows, работающих в защищенном режиме, вы не можете непосредственно вызывать драйвер IPX/SPX или прерыванне NETBIOS, так как эти интерфейсы рассчнтаны на вызов нз реальиого режнма.

Вы можете выйти нз такого затруднительного положения, если воспользуетесь нитерфейсом с защнщенным режимом DPMI (DOS Protected Mode Interface), описанным намн в томе "Библнотеки системного программиста", посвящеином нспользованню защнщеиного режнма.

В рамках нитерфейса DPMI есть функцин, позволяющие из программы, работающей в защищенном режиме, вызывать прерывания илн функцни, предназначеиные для работы в реальиом режиме. Кроме того, в API Windows есть функцин, с помощью которых вы можете заказать для программы защищенного режима буферы, расположенные в первом мегабайте основной оперативной памяти.

Работа в среде Microsoft Windows for Workgroups версии 3.1

Замечания, сделаниые выше, относятся н к Microsoft Windows for Workgroups версни 3.1. Одиако эта операцноиная система может и не поддерживать протоколы

IPX/SPX. С помощью приложения Control Panel вы можете подключить или отключить поддержку сети Novell NetWare и протоколов IPX/SPX.

Если поддержка Novell NetWare не используется, вам доступен протокол NETBIOS, который является "родным" протоколом для Windows for Workgroups.

Мы проверили работу NETBIOS через интерфейс прерывания INT 2Ah при работе в виртуальной машине MS-DOS. Сразу после загрузки сетевых драйверов Windows for Workgroups интерфейс NETBIOS недоступен. Однако в среде виртуальной машины MS-DOS этот интерфейс появляется.

Приведем два фрагмента системы "клиент–сервер", проверенной нами при работе в Windows for Workgroups. Вы сможете найти полные исходные тексты на дискете, которая продается вместе с книгой (эти исходные тексты почти полностью повторяют тексты системы "клиент–сервер", работающей с датаграммами).

Первый фрагмент проверяет присутствие интерфейса NETBIOS:

```
// Проверяем наличие интерфейса NETBIOS
        regs.h.ah = 0;
        int86(0x2a, &regs, &regs);
        errno = 0;
        if(regs.h.ah  == 0) {
             errno=0xff;
        }
```

Второй фрагмент предназначен для вызова NETBIOS через прерывание INT 2Ah:

```
// Функция для вызова NETBIOS
    void NetBios(void) {
        sregs.es = FP_SEG(&ncb);
        regs.x.bx = FP_OFF(&ncb);
        regs.h.ah = 0x4;
        regs.h.al = 0x1;
        int86x(0x2a, &regs, &regs, &sregs);
    }
```

ОГЛАВЛЕНИЕ

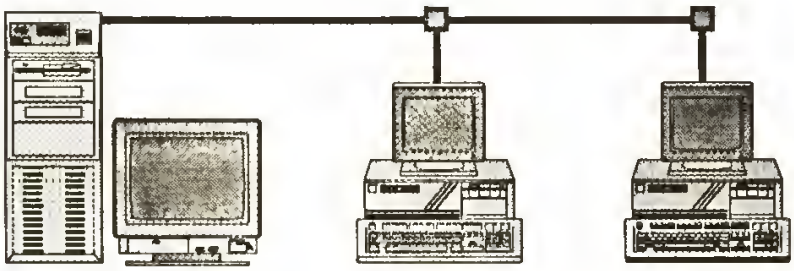
VIMCOM

NOVELL
authorized reseller

ВИМКОМ

Комплексные услуги
по установке компьютерных сетей:

- поставка необходимого оборудования;
- программные продукты фирмы "Novell";
- установка компьютерной сети "под ключ";
- гарантийное обслуживание;
- техническая поддержка.



Вам не нужно быть специалистом в области компьютеров, вместе мы найдем оптимальное соотношение стоимости и производительности оборудования.

Наша цель – сделать каждый проект, в котором мы участвуем, успешным.

Мы ждем Вас!

Россия, 111524, г. Москва
Электродная 10, ВИМКОМ
Телефон: 176-0928, 176-1249,
176-0560, факс: 176-7998.
e–mail: net@infa.msk.su